基于主导值的计算和数据自动划分算法

丁 锐 赵荣彩 韩 林

(解放军信息工程大学计算机科学与技术系 郑州 450002)

摘 要 计算和数据自动划分是并行化编译中一种自动分配计算和数据到各个处理机的优化技术,划分的结果直接影响程序并行的性能。数组是划分处理的主要对象之一,一些数组分布后的收益不高,但带来的并行约束却能对其它数组的划分产生干扰,导致大量数据重分布通信的产生。现有的划分算法中没有约定数组分布的优先次序,因此无法限制这些数组并行约束的传播,降低了优化编译器后端自动生成并行代码的性能。提出了一种基于主导值的计算和数据自动划分算法:将划分过程中数组对程序并行性的影响量化为主导值,并依据主导值的大小约定数组分布的优先次序,限制干扰数组并行约束的传播速度,提高划分结果的合理性。实验结果表明,算法能够获得良好的划分效果。

关键词 自动并行化,计算划分,数据分布,主导值,约束

中图法分类号 TP314 **文献标识码** A

Automatic Computation and Data Decomposition Algorithm Based on Dominant Value

DING Rui ZHAO Rong-cai HAN Lin

(Department of Computer Science and Technology, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract Automatic computation and data decomposition are an optimization technique that distributes computations and data onto different processors. The decomposition result has a direct impact on the performance of program's parallelization. Array is one of main targets of treatment for the decomposition algorithm, and some profits of them are not enough after parallelization, but it will bring constraints and disrupt the other distribution of array, leading to large amounts communication of data re-distribution. The decomposition algorithm in existing has no agreement in the order of array distribution, therefore can't restrict propagation of constraint of array's parallelization, reducing performance of optimized parallel code automatically generated by the back-end compiler. This paper presented an automatic computation and data decomposition based on the dominant values. Algorithm quantified the impact of array on the programs' parallelism as the dominant value, and agreed priorities of distribution based on the size of the dominant values of array, limited the spread speed of constraints of interference array, improved the reasonableness of decomposition results. Experimental results show that the algorithm can get good decomposition results.

Keywords Automatic parallelization, Computation decomposition, Data decomposition, Dominant values, Constraint

1 引言

在分布存储结构的并行系统中,各个计算节点都拥有自己的存储器,节点间需要明确的消息传递来完成数据的交换。由于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,因此在自动并行化过程中如何对串行代码进行正确的并行性分析与优化,发掘其中可识别的并行部分并且将计算与数据合理地自动分配到各个处理机是实现分布存储环境下并行化编译的关键。

并行化编译中的计算和数据划分就是一种将程序中的计算和数据自动分配到各个处理机的优化技术。计算和数据划分由计算划分和数据分布两者构成。其中数据分布主要处理的对象是数组。数组在进行分布时,会产生相应的并行约束,一方面自身不能满足约束就会产生重分布,另一方面又通过循环中计算划分和数据分布的映射关系,约束其它数组的分布选择。

不同的数组对程序的并行性能的影响程度是不同的。一些数组本身适合分布,相反,另一些数组分布后带来的并行约束反而会干扰其它数组和循环的并行,降低并行的收益。因此,编译器对数组进行分布的先后次序,将给自动并行化后程序的并行性能带来截然不同的影响。考虑图1中的例子,该例是NPB3.2.1中BT的compute_rhs函数的简化。

经过分析后能够发现,us、rho_i 的读引用导致其在loop2/3/4的 $i/j/k$ 维无法求出无通信的一致分布。由此产生3种划分结果:1)rhs、us、rho_i 在loop1/2/3/和loop4分别分布 k 维和 i 维,并在2个区域间数据分布不一致,产生数据重组通信;2)rhs、us、rho_i 在loop4也分布 k 维,但需要先通过近邻通信对 us、rho_i 做只读数组复制优化;3)不分布 us、

到稿日期:2011-04-20 返修日期:2011-07-20 本文受"核高基"重大专项子课题(863)(2009AA01220,2009zx10036-001-001)资助。

丁 锐(1984—),男,博士生,主要研究方向为并行编译,E-mail:dingruilele@163.com;赵荣彩(1957—),男,教授,博士生导师,主要研究方向为体系结构、先进编译等;韩 林(1978—),男,博士,讲师,主要研究方向为并行编译。

rho_i(关闭 loop1 的划分)，*rhs* 获得一致分布。

第 3)种划分结果在 4 个循环之间不需要通信，虽然只开发了 *rhs* 数组和后 3 个循环的并行，但相比减少的通信代价，显然是值得的。但是以往的划分算法却很难求出第 3)种结果，原因是都没有约定数组分布的优先次序[1-14]。并行编译器对程序中一些数组的分布后取得的并行收益有限，但产生的约束却对其它数组的合理分布设置了障碍，从而影响到程序整体的自动并行化效果。如何合理地安排数组分布的优先次序，使这样的数组不分布或在其它数组分布过后再进行选择，从而获得更好的划分结果，是本文研究的重点。

```
//loop 1
for i=1,N do
  for j=1,N do
      for k=1,N do
      rho_i(i,j,k)=1.0/u(1,i,j,k)
      us(i,j,k)=u(2,i,j,k)/u(1,i,j,k)
//loop 2
  for i=1,N do
      for j=1,N do
          for k=1,N do
          rhs(1,i,j,k)=rhs(1,i,j,k)+us(i+1,j,k)-rho_i
          (i-1,j,k)+us(i-1,j,k)+rho_i
          (i+1,j,k)
//loop 3
  for i=1,N do
      for j=1,N do
          for k=1,N do
          rhs(1,i,j,k)=rhs(1,i,j,k)+us(i,j-1,k)-rho_i(i,
          j-1,k)+us(i,j+1,k)+rho_i(i,j+1,k)
//loop 4
  for i=1,N do
       for j=1,N do
          for k=1,N do
          rhs(1,i,j,k)=rhs(1,i,j,k)+us(i,j,k+1)-rho_i(i,
          j,k-1)+us(i,j,k-1)+rho_i(i,j,k+1)
```

图 1　程序示例

本文的主要贡献如下：

1)通过数组的存储空间的大小、定值-引用频率的高低以及作用域范围，将数组对程序并行性能的影响地位量化为主导值，并给出了量化算法。

2)提出一种基于主导值的计算和数据自动划分算法，按照数组的主导值由大到小的顺序，约定数组进行分布的先后，依次并行与其相关的循环，限制了低主导值数组的并行约束的传播速度。

本文第 2 节介绍了相关工作；第 3 节分别介绍了主导值的计算和基于主导值的自动划分算法；第 4 节进行了实验结果与分析；最后总结全文。

2　相关工作

Anderson 和 Lam 的算法使用贪婪算法来自动划分计算和数据[1,9,10]，在保证并行性的前提下，逐个合并尽可能多的嵌套循环到同一个静态分解区域中。Xia 采用分解空间融合的方法求解并行分解[4]，Han 则依据融合控制流的层次分解图来逐步合并[3]。他们的算法虽然都开发出了很高的并行度，但均没有体现不同数组在划分算法中的影响程度，因此存在进一步改进的空间。

Lee 在文献[6]中先对每一个循环进行维间对齐，并自底向上的对不需要进行重对齐的区域进行合并。然后对各个无法合并的程序片段求主导数组，通过主导数组的划分迅速影响到片段内计算的划分和其它数组的分布。虽然 Lee 提出了主导数组的概念，但与本文的不同之处在于，他在片段内求出主导数组不是为了约定数组分布的先后次序，而是为了快速在片段内确定分布策略，因为区域内已经对齐。所以其它数组在对齐阶段已经参与分布且添加了约束限制，在确定主导数组的分布策略时无法排除这些约束的影响。

Kremer 等人使用 0-1 整形规划来求解动态的数据分布[2]，但为了避免搜索空间过大，一般只考虑行优先、列优先以及其它受限制的分布策略，因此找到的分布策略有可能是次优的。Lim 提出了一种最大化并行的仿射划分方法[11,12]，但该算法只对计算进行了划分，而未考虑数据的分布，因此算法只适用于共享主存多处理机系统。

Zhang 提出了一种基于代表元的划分算法[5]，用于解决由非紧密嵌套循环、不同语句对数组元素访问步长的不同以及一条语句对数组的多次引用存在重叠而无法求出循环无通信划分方向的问题。虽然通过选取代表元，能够在求无通信的划分方向时去除分布数组产生的部分约束，但无法避免重分布。

Anderson，Beckmann 和 Kremer 分别提出了只读数组复制技术[1,13,14]。只读数组复制技术通过发现只读数组的复制来满足并行处理中固有的读写操作来提高并行度，但只读数组复制是对划分算法的补充优化，不是独立的划分技术，同时其本身大部分需要通信的支持，并且会耗费更多的本地内存。

还有一些研究集中在专一的应用领域。Brian 提出一种算法专门对技术等级的应用程序做自动并行化[7]。Cao 提出一种基于模式匹配来自动并行 2-D 叠前偏移程序的算法[8]。

不同的数组在分布时取得的并行收益是不一样的。上述算法都没有约定数组分布的优先次序，无法在对高收益的数组进行分布时，排除低收益数组分布产生的约束，从而无法求得合理的分布。

3　基于主导值的自动划分算法

3.1　计算数组的主导值

数组能否主导程序并行的性能，一方面看数组数据迁移的代价，另一方面看数组在计算中所占的比重。数组在程序中的主导地位应是其数据量大小、定值-引用的频率以及作用域范围的综合考虑的结果。

数组中存储的数据量大，则数组元素就多，占据的存储空间也就越大。再对这些数组元素进行读写操作时，需要更多的计算才能完成。如果数组发生数据重分布，大量的数据元素无疑会增大数据迁移的代价。因此在考虑数组的主导地位时，数组的数据量大小是必不可少的因素。

数据重分布是由数组定值与引用的分布不一致引起的。程序中一般不会出现只读数组或只写数组，因此数组的定值频率越高，说明数组在程序中越活跃，围绕数组展开的计算也会相应增加。同时定值次数多，也说明数组在程序中多处存在读/写引用，这些引用带来的约束，给数组要求的一致性分布设置了不小的障碍，从而在多处引发重分布通信。因此，需要把数组的定值-引用的频率纳入到数组主导地位的考虑因素中。

```
algorithm Count_Dominant
  (Gc:Gc=(V,E))/ * extended_CFG * /
  for each (PU's Gc) do
    Get first node v in Gc;
    Count_Single_Node(Gc,v,1)
  endeach
end algorithm
algorithm Count_Single_Node
    (Gc:extended_CFG,/ * Gc=(V,E) * /
    v:v∈V
    iter:count of outerloop's iteration)
    if(v is doall)do
      for each array x defined in v do
      if(gen(x)∩out(x)≠∅)do
        Dominant(x)=Dominant(x)+ϕv(x) * |x| * iter
      end foreach
      return
    else
      if(v's firstchild≠NULL)do
          iter=iter * Get_iter(v)
          Count_Single_Node(Gc,v's firstchild,iter)
    endif
    if(v's firstsibling≠NULL) do
      Count_Single_Node(Gc,v's firstsibling,iter)
    endif
  end algorithm
```

图 2　主导值计算算法

数组的作用域范围同样对其主导地位产生影响。如果数组在循环中定值，并在循环中引用，定值的数据没有到达其它循环，那么它就是循环的私有化数组，产生的并行约束其实没有约束力。因此在考虑数组主导地位时，要考虑数组的作用域，将私有化数组排除。为了综合量化数组的这 3 种性质，给出主导值的定义，如图 2 所示。

定义 1　如果数组在循环中定值，又在循环外引用，则将数组的存储空间与定值调用次数的乘积，称之为数组的主导值。

设数组 x 的大小是 $|x|$，x 有 n 个写数组引用的 $gen()\cap out()\neq\emptyset$，$iter(x,i)$ 是 x 第 i 个写引用的外层无关索引的迭代总数，则主导值计算如下：

$$Dominant(x)=\sum_{i=1}^{n}|x| * iter(x,i)$$

其中数组的存储空间能够从其定义处获得，定值-引用对的次数以及作用域则能够通过到达-定值分析获得。数组主导值的具体求解算法如图 2 所示。其中 extended_CFG 是通过区间分析在控制流图的基础上进一步的抽象。其中过程调用蜕化成调用结点，完美嵌套循环蜕化成循环结点，非完美嵌套的循环层蜕化成层结点。结点的同层结点按照控制流连接在一起成为兄弟，深层结点则成为其孩子。

3.2　单嵌套循环的计算与数据划分

仿射分解是一种非常有效的表示和求解数据分布和计算划分的方法。数据分布刻画的是 m 维数组空间到 n 维处理器空间的映射，表示为 $A->P:d(\vec{a})=D\vec{a}+\vec{\delta}$，其中 D 是一个 $n\times m$ 的线性转换矩阵，$\vec{\delta}$ 是偏移常向量，$\vec{a}$ 是数组索引向量。计算划分刻画的是 l 维迭代空间到 n 维处理器空间的映射，表示为 $L->P:c(\vec{i})=C\vec{i}+\vec{\gamma}$，其中 C 是一个 $n\times l$ 的线性转换矩阵，$\vec{\gamma}$ 是偏移常向量，$\vec{i}$ 是迭代向量[1,9,10]。

```
algorithm Single_Node_Decomp
input: p_list,c_list,v,Gc
output :

  Init_Nullspace(p_list,c_list,v)
  Compute_Decomp(p_list,c_list,v)
/ * use cost model to evaluate the benefit of parallelize * /
  if (cost_evaluation)
     commit parallelization of v
     Update_Info(p_list,c_list,v)
  else
     give up parallelization of v
/ * add serial - contraint of array to contraint list * /
     foreach array x in v do
       add(serial_N(Dx),v)to c_list)
     end foreach
  endif
end algorithm

algorithm Update_Info
input : p_list,c_list,v
outpu : p_list,c_list

  / *add array contraint to list * /
     foreach array x in v do
          add (N(Dx),v) to c_list
     end foreach
  / *add new parallelized array to list * /
     foreach array x defined in v & do
       if(x is not in p_list)do
          add x to p_list
     end foreach
     return(p_list,c_list)
end algorithm
```

```
algorithm Init_Nullspace
input : p_list,/ * list of parallelized array * /
        c_list,/ * list of constraint * /
        v:v∈Gc
output : N(Cv), ∪ N(Dx)  (x∈{arrays in v})
/ * Nullspace of computation and data decomposition * /

/ * add doacrosss contraint of loop into Nulllspace * /
     foreach depth q in v do
       if(q is doacorss)do
         / / ēq is q th element vector of iteration space
           add ēq to N(Cv)
     end foreach


/ * add contraints of array defined in v to Nullspace * /
     foreach array x defined in v do
       if(gen(x)∩out(x)≠∅)do
          add range(F^1_xv(i) - F^2_xv(i)) to N(Dx)
       endif
  / / v' is the loop node which contraints of contraints_list belong to
       foreach v' in c_list do
    / /ed is define - use edge of data flow analyse
         if(edx between v and v')do
           add N(Dx)' of v' in c_list to N(Dx)
       end foreach
     end foreach

  / *add contraints of read - only array * /
     foreach read - only array reffence x in v do
       if(x is in p_list)do
          add range(F^1_xv(i) - F^2_xv(i)) to N(Dx)
     end foreach
     return(N(Cv), ∪ N(Dx))  (x∈{arrays in v})
end algorithm
```

图 3　单嵌套循环划分算法

当等式 $D_x(f_{xj}^r(\vec{i}))+\vec{\delta}_x=C_j(\vec{i})+\vec{\gamma}_j$ 成立时，意味着循环迭代及其引用的数组元素被映射到同一个处理器，计算和数据线性对准。此处引入了计算划分矩阵 C_j 和数据分布矩阵 D_x 的核空间，其物理意义是计算和数据需要被指定到相同处理器的空间约束。

在单个嵌套循环内使用该等式，以求解单个嵌套循环内无通信的计算划分与数据分布结果。与前人的不同之处在于，谨慎地选择每个参与分布的数组，控制核空间的增长速度，只将以下 3 类数组的约束添进核空间：已经分布的数组、当前的主导数组以及在循环中定值的非私有化数组。具体算法图 3 所示。

这样就将私有化数组和未并行数组的读引用从循环计算划分和数据分布的核空间中排除，减少了在循环内求无通信解时的约束条件。同时，为了避免被排除的读数组在之后的分布中影响当前循环的已有的划分结果，算法最后计算循环划分对这些数组产生的约束，并添加到全局约束链中。在之后的并行中，约束链中的核空间将影响这些数组分布的选择。

3.3 全局计算和数据自动划分

为了发掘外层粗粒度并行，算法按照抽象流图自顶向下的顺序查找最外层 Doall 的层节点。找到后，内层包含的各类程序结构统一视为语句，外层循环可直接并行。开发外层循环的并行性对于提高粗粒度并行具有重要的作用。

如果找不到层节点的并行，算法按照自底向上的规则进行动态划分，即首先对最内层的节点进行求解，然后逐层向上求解。同时，并行程序的执行依然要遵循串行执行时的控制流顺序，为了减少把不同的静态划分区域夹在中间，算法中引入了程序控制流用以约定同层内嵌套循环间的求解次序。通过约定自底向上和同层的控制流的求解次序，来减少被循环包围的通信。具体的算法如图 4 所示。

```
algorithm Dominat _ Decomp
input: p_list, c_list
       dominat _ array _ list,   //dominat value of arrry
       G_c : G_c = (V, E)    //extended _ CFG
output : decomposition of program

    p _ list = c _ list = NULL
    while ! dominat_array_list is empty do
      //get dominat array
      x = array of Max_Dominat(dominat_array_list)
      if (x is Gobal ) do
        v = first node in G_c of main_pu
        Dominat _ Decomp _ Singlenode(v, x, G_c, plist, clist)
      else
        current_pu = pu which x belongs to
        v = first node in G_c of current_pu
        Dominat _ Decomp _ Singlenode(v, x, G_c, plist, clist)
      endif
      remove(x, dominat_array_list)
      if (all loop is reached)do
        exit
    end while

    return decomposition of program
end algorithm
```

```
algorithm Dominat _ Decomp _ Singlenode
input : v : v ∈ G_c,
        x : dominat array,
        p_list,
        G_c : G_c = (V, E)
output :
  if (v has x || v has array in p_list)do
    if (v has doall )
        Single_Node_Decomp(p_list, c_list, v)
    elseif (v is call)
        current_pu = pu which v calls to
        v' = first node in G_c of current_pu
        //v is call node, process the pu called by v
        Dominat _ Decomp _ Singlenode(v', x, G_c, plist, clist)
    else
        no doall loopnest, give up parallelization of v
        foreach array x in v do
          add(serial_N(D_x), v)to c_list)
        end foreach
    endif
    if (v's firstchild ≠ NULL)do
       Dominat _ Decomp _ Singlenode(v's firstchild, x, G_c, plist, clist)
  endif
  if (v's firstsibling ≠ NULL) do
     Dominat _ Decomp _ Singlenode(v's firstsibling, x, G_c, plist, clist)
  endif
  return
end algorithm
```

图 4　基于主导值的划分算法

算法按照数组的主导值的大小逐步进行并行与相关的循环。在每一轮并行中，先选择主导值最大的数组作为主导数组，然后选择处理与主导数组相关的循环。这样就将不在主导循环中定值的数组暂时排除在并行之外，以延缓它们的核空间对主导数组分布一致性的影响。

4 实验

首先进行实例分析，然后对应用程序进行测试。并将实验结果与 Anderson 和 Han 的算法进行比较，以此说明本文算法的有效性。

Anderson 等率先提出了仿射划分，并使用贪婪算法来自动划分计算和数据。很多学者都在他们的工作基础上进行研究和改进。Han 的研究是项目组已有的工作，他的算法通过融合控制流的层次分解图来逐步合并，能够有效减少分解中数据重分布的次数以及由重分布产生的通信开销。因此，本文选取 Anderson 和 Han 的算法来进行对比。

4.1 实例分析

对基准测试集 NPB3. 2. 1 中 BT 的 compute_rhs 过程进行分析。compute_rhs 中包含 11 个循环，3 维数组 *rho_i*、*us*、*vs*、*ws*、*square* 在 loop1 中定值，其后被引用。4 维数组 *rhs* 在过程的多处进行定值，4 维数组 *u* 是只读数组。例 1 就是 *compute_rhs* 过程的一个缩影。应用 Count_Dominant 算法计算 *compute_rhs* 各个数组的主导值，结果按由大到小的顺序列于表 1。

表 1　数组主导值

Array	Dominant value	Rmark
rhs	$20\times5\times N^3$	defined 20 times
rho_i/us/vs	$1\times N^3$	defined 1 time, in Loop
ws/square/qs	$1\times N^3$	defined 1 time, in Loop
u	0	read-only array
forcing	0	read-only array

rhs 数组在函数的多处进行定值并被引用，本身又是 4 维数组，且作用域没有私有化，因此是 *compute_rhs* 中主导力最

强的数组。*forcing/ u* 在函数中只读，对并行的影响最小。*compue_rhs* 函数与图 1 的结构类似，先不考虑只读数组，*rho_i* 等 6 个数组的主导值小于 *rhs*，但它们读引用产生的并行约束会对 *rhs* 的分布造成影响。

Han 的算法按照控制流会先并行 *rho_i* 等 6 个数组，因此无法将其识别为只读数组，划分结果为图 1 示例的第一种划分结果。Anderson 的算法先选择通信量最大的循环进行并行，能够对 *rho_i* 等数组使用只读数组复制优化，划分结果为图 1 示例的第二种。表 2 中列出了 3 种算法划分后并行的循环和引发的通信。

表 2　划分结果对比

Algorithm	Parallelized loop	Communication	Amount of Communicaiton
this paper	10 of 11	1	$5\times N^3$
Han	11 of 11	8	$16\times N^3$
Anderson	11 of 11	7	$5\times N^3+6(np-1)\times N^2$

文中算法的划分结果为图 1 的第 3 种，从表 2 可以看出，文中算法的划分结果产生的通信从个数和数据量上都是最小的。使用只读数组复制虽然减少了一定的通信量，但是包含不少通信启动时间，耗费同样是巨大的。通过主导值划分算法，虽然并行的循环没有原来多，但是减少了通信。鉴于分布存储结构下通信能力与计算能力的不匹配，这无疑是值得的。

4.2　测试

在 SunWay 集群下对基准测试集 NPB3. 2. 1 中的 BT (Block Tridiagonal)、SP(Scalar Pentadiagonal)和 LU(Lower-Upper symmetric Gauss-Seidel)进行了测试。SunWay 集群由 32 个 3. 2G 双 Nocona 64 位处理器计算结点、2 个监控结点、4 个 I/O 结点和 2 个服务结点组成。

BT 是块状三角的缩写，用于模拟计算流体动力学，测试选取了 A 规模。通过 open64 平台对 BT 做自动并行化，生成并行程序。并通过 SUIF、KAP 平台分别生成 Anderson[1]算法和 Han 算法[3]的并行程序。BT 中 *compute_rhs*、*x_slove*、*y_slove* 和 *z_slove* 是包含在主迭代中被反复调用的核心函数，它们主要用于计算 *rhs* 数组。通过对主导数组 *rhs* 的分布，这些函数包含中的循环都可以被划分，且产生少量的数组重组通信。测试结果如图 5 所示。

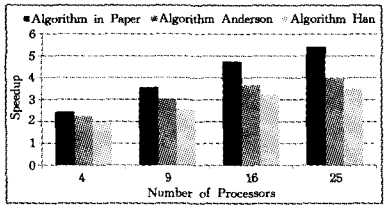

图 5　BT A 规模加速比

SP 是五对角方程的缩写，用于求解 5 对角线方程组。SP 和 BT 的结构类似，但是其主要数组的第一维都不能分布，导致同步通信需要耗费很多时间在本地数据的打包(Pack)和解包上(Unpack)，这影响了程序的性能。测试结果如图 6 所示。

LU 是稀疏矩阵分解时的缩写，基于对称超松弛法求解块稀疏方程组。LU 的 *jacld* 和 *blts* 是核心函数，在主迭代内被反复调用，占据了计算的主要时间，但是这两个函数内包含的循环存在流依赖，自动并行无法对其进行并行化。另一个核心函数 *rhs* 占据的计算时间虽然相比略少，但是围绕着主导数组 *rsd* 的分布，可以划分 *rhs* 中的循环。相比以往的算法，测试结果能够取得较好的加速比。测试结果如图 7 所示。

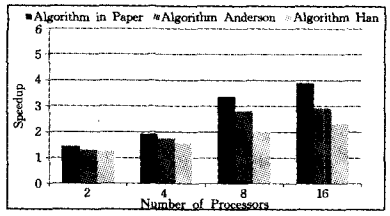

图 6　SP A 规模加速比

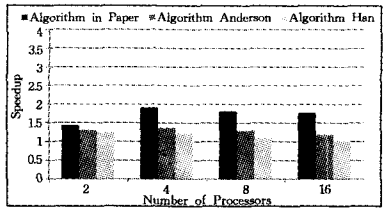

图 7　LU A 规模加速比

实验结果说明本文算法能够有效地减少小主导值数组并行产生的约束，从而更加合理地分布大主导值数组，减少数据重分布的次数和通信量。

结束语　过往的计算和数据划分算法主要把着眼点放在循环上，通过对循环是否并行和划分方式的选择来完成自动划分，往往把数组都设定为了分布。本文从一个新的角度出发，以数组的分布对并行收益的影响为基础，通过主导值为数组的分布排序，在划分与高收益数组相关的循环时暂时关闭低收益数组的分布，限制了其约束的影响范围和传播速度。

主导值的计算主要围绕着数组的写操作，但是当写的次数近似时，读操作的作用就显得尤为重要，例如 NPB 中 CG。因此下一步在改进主导值的计算时，还需要将数组的读操作纳入到考虑的范围。

参 考 文 献

[1] Anderson J M. Automatic computation and data decomposition for multiprocessors [D]. US: Stanford University, 1997

[2] Kennedy K, Kremer U. Automatic Data Layout for Distributed-Memory Machines [J]. ACM Transactions on Programming Languages and Systems, 1998, 20(4): 869-916

[3] 韩林. 面向分布存储结构的并行分解一致性优化技术研究[D]. 郑州：解放军信息工程大学，2008

[4] 夏军，杨学军. 基于数据空间融合的全局计算与数据划分方法[J]. 软件学报，2004，15(09)：1311-1327

[5] 张为华，王鹏，臧斌宇，等. 一种基于代表元的划分算法[J]. 计算机学报，2008，31(3)：400-410

[6] Lee Pei-zong. Automatic data and computation decomposition on distributed memory parallel computers[J]. ACM Transactions on Programming Languages and Systems, 2002, 24(1): 1-50

[7] Brian S A. Enabling automatic parallelization of industrial-grade applications [D]. US: Purdue University, 2010

[8] Zhen Cao, Yuan Dong, Wang Sheng-yuan. Domain-specific pattern matching based automatic parallelization demonstrated by 2-D prestack migration: Parallel and Distributed Systems (ICPADS)[C]// 15th International Conference. Shenzhen, China, 2009: 973-980

(下转第 303 页)

和重叠性。

因此通过异步执行能够使 GPU 的执行单元和存储器单元同时工作，提高资源利用率，这样可以在一定程度上减轻由于大量数据拷贝带来的计算效率的降低。

在第一阶段和第二阶段更新数据块矩阵时，计算数据量较小，不适合异步处理。而主要的数据量集中在第三阶段，并且注意到该部分最短路径计算更新不存在相互之间的先后顺序，因此可以采用异步处理，异步并行地进行计算和数据的拷贝，从而有效地隐藏数据交换带来的延迟。

同时，为了进一步提高通信效率，根据第一阶段和第二阶段数据量比较小的特点，第一阶段和第二阶段都在全局内存中操作，并不拷贝出来，只是在第三阶段时，才将大量的数据在内存和显存直接分配拷贝和处理。

表 2 给出了部分大规模数据矩阵的测试结果。最后在矩阵规模 $N=15360$，即矩阵大小为 15360×15360 时，数据量已经到达 1.5625G，这对单 PC 来说已经是很大的数据量，所以可以验证算法具有较强的扩展性和实用性。

表 2 大规模数据矩阵下的加速比

矩阵规模 N	CPU 端 Floyd	GPU 端 Floyd	加速比	GPU 端流处理 Floyd	加速比
	时间/s	时间/s		时间/s	
11264	6125.262	405.054	15.12	244.7	25.03
12288	7952.256	515.83	15.41	291.83	27.35
13312	10110.59	631.91	16.23	367.1	27.54
14336	12627.89	774.71	16.35	448.88	28.15
15360	15531.75	919.038	16.9	533.92	29.09

如表 2 所列，在使用异步并行处理之前，由于浪费了大量的通信时间，只有 15 倍左右的加速。而使用了流处理后，节省了一部分通信时间，可以达到 25 倍左右的加速。

结束语 针对全源最短路径问题的两种串行算法，本文提出一种基于 GPU 的并行算法。此并行算法实现简单，能够处理不同规模的数据并且在大规模并行计算上有着广泛的应用前景。实验结果表明，此并行算法降低了计算所有点对之间最短路径问题的时间，提高了计算效率，实现了高达 155 倍的并行加速比。

算法的不足之处在于如果数据矩阵规模过大，则算法的执行效率稍低，通信开销过大。如何克服这个缺陷，也是今后需要解决的问题。同时，将算法进一步扩展到单 PC 多显卡和 GPU 集群上，也是一项重要的研究内容。

参 考 文 献

[1] Floyd R W. Algorithm 97(SHORTEST PATH)[J]. Communications of the ACM, 1962

[2] Lawler E L. Combinatorial Optimization: Networks and Matrodis[M]// Holt, Rinehart and Winston, 1976

[3] Harish P, Narayanan P J. Accelerating large graph algorithms on the GPU using CUDA[C]// Proc. 14th Int'l Conf. High Performance Computing (HiPC'07). Dec. 2007: 197-208

[4] Okuyama T, Ino F, Hagihara K. A Task Parallel Algorithm for Computing the Costs of All-pairs Shortest Paths on the CUDA Compatible GPU[C]// Proceedings of 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications. IEEE, 2008: 284-291

[5] Katz G J, Kider J T, Jr. All-pairs shortest-paths for large graphs on the GPU[C]// Proc. of the 23rd ACM

[6] 周益民，孙世新. 一种实用的所有点对之间最短路径并行算法[J]. 计算机应用，2005，25(12)：2911-2913

[7] Cormen T H, Leiserson C E, Rivest R L, et al. Introduction to Algorithms(Second Edition)[M]. The MIT Press, 2001

[8] Breshears C. The Art of Concurrency[M]. O'Reilly Media, Inc, 2001

[9] 张树，褚艳利. 高性能运算之 CUDA[M]. 北京：中国水利出版社，2009

[10] 卢凤顺，宋君强，银福康，等. CPU/GPU 协同并行计算研究综述[J]. 计算机科学，2011，38(3)：5-9

[11] 唐策善，李龙澍，黄刘生. 数据结构—用 C 语言描述[M]. 北京：高等教育出版社，2000

[12] 胡文美. 大规模并行处理器编程实战[M]. 北京：清华大学出版社，2010

[13] Tanenbaum A S, Woodhull A S. Operating Systems Design and Implementation[M]. Englewood Cliffs, NJ: Prentice-Hall International Inc, 1997

(上接第 294 页)

[9] Anderson J M, Lam M S. Global optimizations for parallelism and locality on scalable parallel machines[C]// Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation. Albuquerque, New Mexico, USA, 1993: 112-125

[10] Anderson J M, Lam M S. Data and computation transformations for multiprocessors[C]// Proceedings of the 5th ACM/SIGPLAN Symposium on Principles and Practice of Parallel Programming. Santa Barbara, California, USA, 1995: 166-178

[11] Lim A W, Lam M S. Maximizing parallelism and minimizing synchronization with affine transforms[C]// Proceedings of the Conference Record of the 24th ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages. Paris, France, 1997: 201-214

[12] Lim A W, Lam M S. Maximizing parallelism and minimizing synchronization with affine partitions[J]. Parallel Computing, 1998, 24(3/4): 445-475

[13] Ulrich K. Automatic data layout with read-only replication and memory constraints[C]// Proceedings of the 10th international Workshop on Languages and Compilers for Parallel Computing. Chapel Hill, NC, USA, 1998: 419-422

[14] Olav B, Jpaul H. A liner algebra formulation for optimizing replication in data parallel programs[C]// Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing. Youktown Heights, NY, USA, 2000: 100-116