在 PVM 应用程序中调用 ScaLAPACK 库函数方法[*1)]

张云泉　迟学斌

（中国科学院软件所并行软件研究开发中心）

METHOD OF CALLING SCALAPACK IN PVM APPLICATIONS

Zhang Yun-quan　Chi Xue-bin

(*R & D Center for Parallel Software, Institute of Software, Chinese Academy of Sciences*)

Abstract

As the parallel version of LAPACK, ScaLAPACK will provide convenient and powerful parallel programming platform for parallel computing society. In this paper, we propose a PVM based ScaLAPACK application programming structure. Based on this general programming structure, we give three initialization methods of BLACS that can be used under different situations. The data distribution fashion used in ScaLAPACK is also introduced to help the user to understand them better. For those who are familiar with PVM parallel programming, they can call ScaLAPACK subroutines directly using our methods, without concerning ScaLAPACK details too much. Finally, several shortcomings of current version ScaLAPACK package are discussed and an API is proposed to improve its usability.

§1. 引　　言

ScaLAPACK 是 Scalable Linear Algebra PACKage 的缩写，是为在基于消息传递的 MIMD 并行计算机系统上解数值线性代数问题，并由美国橡树岭国家实验室和田纳西大学等联合开发. 它支持对 (1) 线性代数方程组问题 (2) 最小二乘问题 (3) 特征值问题 (4) 奇异值分解等问题的求解 (参见文献 [1—3]). 这些问题由于在科学与工程计算中经常出现，它们的高效求解成了应用程序获得高性能的关键. 随着计算机的发展，相继开发了 Linpack, Eispack, LAPACK 和 ScaLAPACK 等数值软件包，利用这些软件包能节省大量的应用程序编写时间.

* 1998 年 3 月 9 日收到.

1) 本工作得到国家攀登 B 计划项目、国家 863 高技术计划项目和国家自然科学基金项目的支持.

作为在科学计算中广为使用的 LAPACK 软件包的并行版， ScaLAPACK 是面向分布式并行计算机的数学软件包，而 BLAS 和 BLACS[4] 是 ScaLAPACK 的两个基础软件包：其中 BLAS 参与单处理器的浮点计算，BLACS 是 ScaLAPACK 的通信界面. 可以说， BLAS 决定 ScaLAPACK 的计算速度， BLACS 决定 ScaLAPACK 的通信速度. 用户若想调用 ScaLAPACK 的库函数，就必须了解 BLACS 通信函数库的使用方法和相关的 ScaLAPACK 的新知识 (如数据分布方式等).

但是，用户一般不愿意抛弃已有的并行环境 (比如 PVM) 去学习另一个全新的并行环境，如 BLACS 等. 一个原因是用户在原有环境中已积累了大量的编程经验和成熟的程序代码，另一个原因是学习新的并行环境需要大量的时间，同时由于并行环境发展太快，用户一般不愿为一个很可能马上过时的新环境付出太多的时间. 为了让用户及时利用并行数学软件库 ScaLAPACK 的最新成果而又不抛弃已有的成熟的 PVM 并行环境，本文着重解决从 PVM 直接调用 ScaLAPACK 库函数的问题，并围绕这一问题对 ScaLAPACK 的数据分布方法作了简要的介绍，并给出一个基于 PVM 的 ScaLAPACK 库函数调用的例子以便于理解. 最后，根据当前 ScaLAPACK 软件包存在的一些问题，从方便用户使用的角度提出了为 ScaLAPACK 软件包增加应用程序编程接口 (API) 的设想.

§2. ScaLAPACK 库函数的一般调用框架

PVM (Parallel Virtual Machine)[6] 是基于消息传递机制的并行编程环境. 它支持两种编程模型，主从模型和结点模型. 在结点模型中，最常用的风格是 SPMD (Single Program Multiple Data). ScaLAPACK 中所有的库函数都采用 SPMD 模型编写.

为了调用 ScaLAPACK 的库函数 [5]. 最好的方式是调用 BLACS[4] 的函数来生成子进程并初始化 BLACS 并行环境，当然也可以先由用户调用 PVM 函数生成子进程，再按下面介绍的方法初始化 BLACS 并行环境. 为了调用 ScaLAPACK 库函数， PVM 应用程序的一般结构可如下所示：

1. 初始化 PVM 并行环境；
2. 基于 PVM 的应用程序 (无 ScaLAPACK 调用部分)；
3. 初始化 BLACS 并行环境；
4. 按 ScaLAPACK 要求分配矩阵数据；
5. 调用 ScaLAPACK 库函数；
6. 退出 BLACS 并行环境；
7. 基于 PVM 的应用程序 (无 ScaLAPACK 调用部分)；
8. 退出 PVM 并行环境.

其中的第 3—6 步可以在应用程序中多次出现. 本文假定用户对 PVM 编程已经比较熟悉，重点解决第 3—6 步中的问题. 下面将根据上述的一般框架分别介绍 BLACS 及其初始化方法， ScaLAPACK 的数据分配方式及退出 BLACS 的方法.

§3. 基于 PVM 的 BLACS

本节先给出BLACS通信子程序包的简要介绍,然后讨论为在PVM中进行ScaLAPACK库函数调用初始化BLACS并行环境的三种方法.

3.1 BLACS 简介

BLACS(基本线性代数通讯子程序包) 是 ScaLAPACK 库函数和消息传递库 (诸如 PVM, MPI 等) 间的调用界面. 为了动态内存分配的方便, 一般用 C 语言写. BLACS 包含所有 PBLAS(并行版 BLAS) 和 ScaLAPACK 并行实现所需要的通讯函数. 开发 BLACS 软件包的目的是:

- 易于编程. BLACS 将尽可能简化消息传递函数的调用以减少编程错误.
- 易于使用. BLACS 的编程界面和函数功能将易于被数值线性代数方面的程序设计者接受并满足他们的需要.
- 可移植性. BLACS 所提供的界面必须能被大多数的包括异构并行机在内的并行计算机支持.

BLACS 是专为编写求解数值线性代数问题的并行程序而设计的, 也许并不适于其它领域应用程序的编写. 但其全新的设计思想可以被其它领域的应用程序设计者借鉴. 关于 BLACS 的详细资料可参考 [4].

3.2 初始化 BLACS 的三种方法

根据我们使用 BLACS 的经验, 用户可以有三种方法初始化 BLACS, 我们分别称作: 标准方式, 用户生成静态方式和用户生成动态方式. 其中用户生成静态方式和用户生成动态方式适用于从 PVM 直接调 ScaLAPACK 库函数的用户, 并且已经通过实际应用程序的试验.

3.2.1 标准方式

标准方式可用来建立 ScaLAPACK 并行环境和虚的进程网格. 在此过程中, 用户不必显示调用任何 PVM 函数, 它适用于用户完全依赖 ScaLAPACK 进行并行计算的情形. 标准方式的一般过程如下:

1. 所有进程调用 BLACS_SETUP 子程序生成用户指定数目的进程.

2. 调用 BLACS_GET 把 CONTEXT 句柄设为 NOTINCONTXT 以便告诉 BLACS_GRIDMAP 子程序该进程网格需要一个新的 CONTEXT 句柄.

3. 调用 BLACS_GRIDINIT 初始化一个虚的进程网格. 用户可以选择进程的索引顺序 (行为主或列为主) 和该网格的形状 (行数及列数). 但是用标准方式生成的进程网格是最简单的, 用户无法控制进程到网格坐标的映射.

4. 最后按二维块循环方式初始化矩阵数据并调用 ScaLAPACK 函数.

3.2.2 用户生成方式

用户生成方式允许用户自己调用 PVM 函数生成并行子进程并在 PVM 环境下进行计算, 而只在必要时才调用 ScaLAPACK 库函数进行并行计算.

1. 调用 PVMFSPAWN 生成所需要的子进程. 为了初始化 BLACS 并行环境, 必

须用一个数组保存所有参加并行计算的进程任务 ID, 以便调用 BLACS 的 setpvmtids 子程序设定 pvmtid00 全局数组的值.

2. 在 PVM 环境下进行并行计算. 这时用户程序中不必调用 ScaLAPACK 库函数的部分. 该部分的计算结果可以作为 ScaLAPACK 子程序的输入数据.

3. 调用 SETPVMTIDS 初始化 BLACS 的全局数组 pvmtid00. 该数组在进程网格生成, 重映射及进程坐标管理方面有重要作用.

4. 调用 BLACS_GET 初始化 CONTEXT 句柄为缺省值 NOTINCONTEXT. 该句柄对于确定一进程是否在某进程网格中非常有用. 建议用户不要直接对 CONTEXT 赋值, BLACS_GRIDINIT 会为每一网格分配唯一的句柄.

5. 初始化进程网格

• 用户生成静态方式 - 调用 BLACS_GRIDINIT 初始化一个虚的进程网格.

• 用户生成动态方式 - 调用 BLACS_GRIDMAP 按用户指定的进程顺序和形状生成虚的进程网格. 实际上, BLACS_GRIDMAP 与 BLACS_GRIDINIT 功能相似, 但更灵活. 用户可以通过对任务 ID 数组 USERMAP 的赋值来指定进程的网格坐标. BLACS 函数会根据进程在该数组中的顺序来决定进程到网格坐标的映射. 当前 BLACS_GRIDMAP 存在的一个缺点是: 它只搜索 USERMAP 数组的前 P 个元素 (P 是网格中的进程数). 对于不在前 P 个之内的进程会被忽略, 这就要求用户必须把包含在网格中的进程放在 USERMAP 的前 P 个元素里.

6. 数据初始化或重分配. 由于 ScaLAPACK 采用二维块循环数据分配方式, 用户在调用 ScaLAPACK 库函数前, 必须把数据按这种方式准备好.

7. 调用 ScaLAPACK 库函数.

3.3 退出 BLACS

在 ScaLAPACK 中完成并行计算后, 用户想退出 BLACS 环境, 结束计算或在 PVM 环境下继续进行其它计算, BLACS 为此提供了 BLACS_EXIT 子程序.

BLACS_EXIT 首先释放 BLACS 的 CONTEXT 句柄及进程网格, 然后释放 PVM 的发送缓冲区及 BLACS 缓冲区. 它允许用户选择是否退出 PVM 并行环境. 最后, BLACS_EXIT 会把 BLACS 的所有全局变量重置. 这些变量对用户重新进入 BLACS 环境是很重要的.

§4. ScaLAPACK 的数据分配

用户在初始化完 BLACS 并行环境后, 下一步就是按 ScaLAPACK 的数据分布方式重新分配矩阵数据, 否则, ScaLAPACK 的库函数不能正常工作.

4.1 二维块循环数据分配方式

对于为分布式存储并行机设计的并行程序, 数据分配方式的选择是一个基本的且很重要的问题, 它直接关系到所设计的并行程序的性能和可扩展性. ScaLAPACK 采

用二维块循环数据分配方式 (2-D Block Cyclic Data Distribution)[7], 从而保证用 ScaLAPACK 库函数实现的应用程序既能充分利用缓存, 又能有良好的负载均衡.

我们可以这样表述数据分配问题:

对于长度为 n 的向量 V 在 P 个处理器上的分配, 先把 P 个处理器编号为 $0,1,\cdots,P-1$, 然后把向量的第 i 个元素 v_i $(0 \le i < n)$ 分给第 j 个处理器 P_j $(0 \le j < P)$ 且该元素连续存储在该处理器本地数组的第 k 个位置.

因此, 可把向量在处理器上的分配问题看成全局向量下标 i 到下标对 (j,k) 的映射, 其中 j 是处理器序号, k 是本地数组下标. 通过张量积的方式可以把一维向量的映射扩展到二维矩阵的情形.

现在常用的数据分配方式有两种: (1) 块方式 (Block) (2) 循环或卷帘方式 (Wrapped). 其中块方式用于计算量均匀分布在一矩阵上的并行算法, 而循环方式用于计算量非均匀分布在一矩阵上的并行算法. 根据上面的叙述, 块分配方式可以描述如下: 向量 V 的第 i 个元素 v_i 将分配到第 $\lfloor i/L \rfloor$ 个处理器的本地数组的第 $i \bmod L$ 个位置上, 其中 $\lfloor x \rfloor$ 表示比 x 小的最大整数, $L = \lfloor (n-1)/P \rfloor + 1$, 用映射的形式可如下表示:

$$f_1(i) = (\lfloor i/L \rfloor, i \bmod L)$$

很容易的, 循环方式可如下表示:

$$f_2(i) = (i \bmod P, \lfloor i/P \rfloor)$$

其中 P 是处理器数.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}	$a_{1,10}$	$a_{1,11}$	$a_{1,12}$	$a_{1,13}$
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}	a_{29}	$a_{2,10}$	$a_{2,11}$	$a_{2,12}$	$a_{2,13}$
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}	a_{39}	$a_{3,10}$	$a_{3,11}$	$a_{3,12}$	$a_{3,13}$
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}	a_{49}	$a_{4,10}$	$a_{4,11}$	$a_{4,12}$	$a_{4,13}$
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	a_{58}	a_{59}	$a_{5,10}$	$a_{5,11}$	$a_{5,12}$	$a_{5,13}$
a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}	a_{68}	a_{69}	$a_{6,10}$	$a_{6,11}$	$a_{6,12}$	$a_{6,13}$
a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}	a_{78}	a_{79}	$a_{7,10}$	$a_{7,11}$	$a_{7,12}$	$a_{7,13}$
a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}	a_{87}	a_{88}	a_{89}	$a_{8,10}$	$a_{8,11}$	$a_{8,12}$	$a_{8,13}$
a_{91}	a_{92}	a_{93}	a_{94}	a_{95}	a_{96}	a_{97}	a_{98}	a_{99}	$a_{9,10}$	$a_{9,11}$	$a_{9,12}$	$a_{9,13}$
$a_{10,1}$	$a_{10,2}$	$a_{10,3}$	$a_{10,4}$	$a_{10,5}$	$a_{10,6}$	$a_{10,7}$	$a_{10,8}$	$a_{10,9}$	$a_{10,10}$	$a_{10,11}$	$a_{10,12}$	$a_{10,13}$
$a_{11,1}$	$a_{11,2}$	$a_{11,3}$	$a_{11,4}$	$a_{11,5}$	$a_{11,6}$	$a_{11,7}$	$a_{11,8}$	$a_{11,9}$	$a_{11,10}$	$a_{11,11}$	$a_{11,12}$	$a_{11,13}$

图 4.1 分成 4×5 阶的块矩阵 (块大小 3×3)

如果把循环方式中的单个元素换成一个连续块 (r 个元素), 则可得到块循环方式. 此时, 必须先把向量按长度 r 分块, 再把这些块按循环方式分配到处理器上. 用户所关心的除了一块数据分配到的处理器位置 j, 该块数据在本地数组的位置 k, 还有某个数据在它所属的块中的相对位置 t. 这时的映射为

$$f_3(i) = (j,k,t) = \left(\left\lfloor \frac{i \bmod T}{r} \right\rfloor, \left\lfloor \frac{i}{T} \right\rfloor, (i \bmod T) \bmod r \right),$$

其中 $T = rP$[8].

4.2 数据分布的例子

下面，我们通过一个例子来说明 ScaLAPACK 中矩阵的划分和相应的到处理器网格上的映射. 假定矩阵 $A \in R^{m\times n}$ 是 $m \times n$, 行列的块大小分别为 m_b 和 n_b, 因此该矩阵被分成 $\bar{m} = \lceil m/m_b \rceil$ 乘 $\bar{n} = \lceil n/n_b \rceil$ 的块矩阵，其中 $\lceil x \rceil$ 表示比 x 大的最小整数. 设 $m = 11$, $n = 13$, $mb = 3$, $nb = 3$, 则原矩阵分成 4×5 阶的块矩阵，如图 4.1 所示. 如果分布到一个 2×2 的处理器网格上，其结果如图 4.2 所示.

		0						1					
0	a_{11}	a_{12}	a_{13}	a_{17}	a_{18}	a_{19}	$a_{1,13}$	a_{14}	a_{15}	a_{16}	$a_{1,10}$	$a_{1,11}$	$a_{1,12}$
	a_{21}	a_{22}	a_{23}	a_{27}	a_{28}	a_{29}	$a_{2,13}$	a_{24}	a_{25}	a_{26}	$a_{2,10}$	$a_{2,11}$	$a_{2,12}$
	a_{31}	a_{32}	a_{33}	a_{37}	a_{38}	a_{39}	$a_{3,13}$	a_{34}	a_{35}	a_{36}	$a_{3,10}$	$a_{3,11}$	$a_{3,12}$
	a_{71}	a_{72}	a_{73}	a_{77}	a_{78}	a_{79}	$a_{7,13}$	a_{74}	a_{75}	a_{76}	$a_{7,10}$	$a_{7,11}$	$a_{7,12}$
	a_{81}	a_{82}	a_{83}	a_{87}	a_{88}	a_{89}	$a_{8,13}$	a_{84}	a_{85}	a_{86}	$a_{8,10}$	$a_{8,11}$	$a_{8,12}$
	a_{91}	a_{92}	a_{93}	a_{97}	a_{98}	a_{99}	$a_{9,13}$	a_{94}	a_{95}	a_{96}	$a_{9,10}$	$a_{9,11}$	$a_{9,12}$
1	a_{41}	a_{42}	a_{43}	a_{47}	a_{48}	a_{49}	$a_{4,13}$	a_{44}	a_{45}	a_{46}	$a_{4,10}$	$a_{4,11}$	$a_{4,12}$
	a_{51}	a_{52}	a_{53}	a_{57}	a_{58}	a_{59}	$a_{5,13}$	a_{54}	a_{55}	a_{56}	$a_{5,10}$	$a_{5,11}$	$a_{5,12}$
	a_{61}	a_{62}	a_{63}	a_{67}	a_{68}	a_{69}	$a_{6,13}$	a_{64}	a_{65}	a_{66}	$a_{6,10}$	$a_{6,11}$	$a_{6,12}$
	$a_{10,1}$	$a_{10,2}$	$a_{10,3}$	$a_{10,7}$	$a_{10,8}$	$a_{10,9}$	$a_{10,13}$	$a_{10,4}$	$a_{10,5}$	$a_{10,6}$	$a_{10,10}$	$a_{10,11}$	$a_{10,12}$
	$a_{11,1}$	$a_{11,2}$	$a_{11,3}$	$a_{11,7}$	$a_{11,8}$	$a_{11,9}$	$a_{11,13}$	$a_{11,4}$	$a_{11,5}$	$a_{11,6}$	$a_{11,10}$	$a_{11,11}$	$a_{11,12}$

图 4.2　把分割后的块矩阵映射到 2×2 处理器网格上

§5. 调用 ScaLAPACK 库函数示例

本节用一个示意性的例子来说明基于 PVM 的 ScaLAPACK 库函数调用的方法，该程序使用前面介绍的用户生成静态方式初始化 BLACS 并行环境. 关于其它两种方法用户可以按前面介绍的做. 该程序通过调用 ScaLAPACK 的库函数 PDGETRF () 和 PDGETRS () 来求解线性代数方程组 $Ax = b$.

在这个例子中，$a_{ij} = \sum_{k=1}^{\min\{i,j\}} 1/(i-k+1)$, 假设最后方程组的解 $x_i = 0$ 如果 i 是偶数，$x_i = 1$ 如果 $i-1$ 是 4 的倍数，其它情况 $x_i = -1$. 按着这种要求，我们给出一个具体使用 ScaLAPACK 求解此线性代数方程组的程序. 通过此程序可以看出，使用 ScaLAPACK 需要预先确定处理机网格，根据处理机网格形成所需要的数据 (可以在调用 BLACS 有关函数之前进行), 本例使用 2×2 处理机网格求解 11×11 阶线性代数方程组.

```
      PROGRAM pvmsls
      ......................
*     Parameter, Variable declarations and External statements omitted
      NPROW=2
```

```
      NPCOL=2
      np=NPROW*NPCOL
      M=11
      NB=2
      LDA=(M+nb)/nprow+nb
      name='pvmsls'
      me=-1
*   Enroll in pvm
      call pvmfmytid (mytid)
      call pvmfparent (tids(0))
      if (tids (0) .lt. 0) then
        me=0
        tids (0)=mytid
        call pvmfspawn (name, 0, '*', np-1, tids (1), info)
*        ------------------------------------------
*   Send tids array to children for setpvmtids () calling
*        ------------------------------------------
        call pvmfinitsend (0, info)
        call pvmfpack (INTEGER4,np, 1, 1, info)
        call pvmfpack (INTEGER4, tids, np, 1, info)
        call pvmfmcast (np-1, tids(1), 0, info)
      else
        call pvmfrecv (tids(0), 0, info)
        call pvmfunpack (INTEGER4, np, 1, 1, info)
        call pvmfunpack (INTEGER4, tids, np, 1, info)
        do 10 i=1, np-1
          if (mytid. eq. tids (i)) then
            me=i
            goto 20
          endif
10      continue
20    continue
      endif
*     User can do some PVM base parallel computation in this segment.
      ..........................................
*     Begin to initialize BLACS
      call setpvmtids (nprocs,tids)
      CALL BLACS_GET (-1, 0, ICTXT)
      CALL BLACS_GRIDINIT (ICTXT, 'Row-major', NPROW, NPCOL)
*     End of BLACS initialization, you cal call any ScaLAPACK routines now
```

```
*       Set matrix initial information
        CALL DESCINIT (DESCA, M, M, NB, NB, 0, 0, ICTXT, LDA, INFO)
*       Setting values of matrix
*       Matrix values are Aij=sum 1/(i-K+1) (1≤k ≤ 1, l=min (i, j))
        CALL BLACS_GRIDINFO (ICTXT, NPROW, NPCOL, MYROW, MYCOL)
        call ainitmtx (m, nb, a, lda, myrow, mycol, nprow, npcol)
*       ScaLAPACK routine for LU factorization
        CALL PDGETRF (M, M, A, 1, 1, DESCA, IPVT, INFO)
*       Generate the right hand side of linear equations
        CALL DESCINIT (DESCB, M, 1, NB, 1, 0, 0, ICTXT, LDA, INFO)
        call binitvct (m, nb, b, myrow, mycol, nprow)
*       Solving the triangular systems
        call PDGETRS ('n', m, 1, A, 1, 1, DESCA, IPVT, B,
                      1, 1, DESCB, INFO)
*       End of ScaLAPACK routines calling
*       Begin to finalize the BLACS environment
        CALL BLACS_GRIDEXIT (ICTXT)
*     Set parameter to 1 to keep PVM process alive, 0 kill them
        CALL BLACS_EXIT (1)
*       User can do some PVM based parallel computation in this segment.
        ..........................................
*       Finish the PVM process
        pvmfexit (info)
        STOP
        END
```

以上程序先在 PVM 环境下进行初始化和必要的计算 (未给出), 再用前面介绍的用户生成静态方式初始化 BLACS 并行环境, 为 ScaLAPACK 库函数的调用做好准备; 接着, 通过调用 ScaLAPACK 的库函数 PDGETRF 对矩阵 A 进行 LU 分解, 接下来调用库函数 PDGETRS 进行求解. 在退出 BLACS 环境之后, 继续在 PVM 下进行其它的并行计算 (未给出), 最后退出 PVM, 结束所有的并行计算任务.

本例中矩阵及右端项的初始化是通过调用 ainitmtx() 和 binitvct() 实现的. 在熟悉了 ScaLAPACK 的矩阵分布方式之后, 用户可以编写自己的矩阵和向量初始化子程序. 如果读者对完整的程序感兴趣, 可向作者索取.

§6. 结束语

ScaLAPACK 从总体上实现了把共享存储版本的 LAPACK 软件包移植到分布式 MIMD 并行机上的设计构想, 为分布式 MIMD 并行机上的用户和并行程序员提供了

一个强大的并行数值计算编程平台，极大地推动了分布式 MIMD 并行机的推广和应用.

但是，在为日立第二代 MPP 机 SR2201 移植和加速 ScaLAPACK 软件包的过程中，我们发现当前的 ScaLAPACK 作为一个并行数值软件包，对应用程序编程接口 (API) 的设计和支持是很不够的，表现为以下几点：

1. 对数据分布的细节隐藏不够，用户需把过多的精力放在对数据分布的操作上，参数行过长且容易出错；

2. 工作数组放在了参数行，需用户确定而不是程序自动确定，这当然是由 FORTRAN 的静态内存分配决定的；

3. 没有考虑和其它并行环境的接口，如直接从 PVM 调 ScaLAPACK 库函数的问题；

4. 对数值计算的应用程序实现提供的库函数太少，如对矩阵的初始化，计算结果的处理，结果显示等方面.

我们认为，可以通过为 ScaLAPACK 设计一个 C 语言的 API 来解决上述问题，关于该 API 的详细信息参见 [9].

1. 通过把矩阵的数据和分布信息放在一个对象中，把分布信息设为缺省值来减少参数行的长度和对数据分布细节的过多投入，这样，一个参数就可以传递较多的信息，而用户在必要时则可通过调用专门编写的子程序对缺省值进行修改；

2. 由于采用了 C 语言，可以结合具体的子程序，自动分配所需的内存而无需用户的干涉，从而可把工作数组从参数行省掉；

3. 通过编写新的子程序来解决上述第四点中提到的问题.

我们相信通过为 ScaLAPACK 增加新的从应用编程角度设计的 API，不仅可以方便完全采用 ScaLAPACK 编程方法的用户，对于从其它并行环境调 ScaLAPACK 库函数的用户也会极为方便.

参 考 文 献

[1] G. Golub, C. van Loan, Matrix Computation, The Johns Hopkins University Pres, 1996.

[2] J.W. Demmel, Berkeley Lecture Notes on Numerical Linear Algebra, Volume I, 1995.

[3] J.H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press, 1965.

[4] R. Clint Whaley, Installing and Testing the BLACS, October 3, 1995. from Internet

[5] J. Choi, *et al.*, Installation Guide for ScaLAPACK, February 28, 1995. from Internet

[6] Al Geist, *et al.*, PVM 3 User's Guide and Reference Manual, Sept., MIT Press, Cambridge, MA, 1994.

[7] J.J. Dongarra and D.W. Walker, Software Libraries for Linear Algebra Computations on High Performance Computers, *SIAM Review*, **37** (1995), 151–180.

[8] Jack Dongarra, Robert van de Geijin, David Walker, A Look at Scalable Dense Linear Algebra libraries. University of Tennessee, Technical Report CS-92-156, May 1992 (Also LAPACK Working Notes #43).

[9] 张云泉， Adding API to ScaLAPACK Package, 中科院软件所并行软件研究开发中心， 1997 年年报.