МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М. В. ЛОМОНОСОВА

Научно-исследовательский вычислительный центр

О. Б. Арушанян, Н. И. Волченскова

РЕШЕНИЕ ЛИНЕЙНОЙ АЛГЕБРАИЧЕСКОЙ ПРОБЛЕМЫ СОБСТВЕННЫХ ЗНАЧЕНИЙ ДЛЯ СИММЕТРИЧНЫХ И ЭРМИТОВЫХ МАТРИЦ НА ОСНОВЕ ПАКЕТА ScaLAPACK

Учебное пособие

Москва, 2016

О.Б. Арушанян, Н.И. Волченскова

Решение линейной алгебраической проблемы собственных значений для симметричных и эрмитовых матриц на основе пакета ScaLAPACK

(Учебное пособие)

Оглавление

1. Введение.

Уровень и сложность решаемых в настоящее время научных и производственных задач таков, что, как правило, требуется использование высокопроизводительных суперкомпьютеров, в частности с распределенной памятью. Подготовка специалистов в этой области в МГУ им. М.В. Ломоносова ведется на базе Суперкомпьютерного центра (СКЦ МГУ), состоящего в настоящее время из двух высокопроизводительных суперкомпьютеров: СКИФ МГУ "Чебышев" (60 Тфлоп/с) и "Ломоносов" (1,7 Пфлоп/с).

Описываемый в настоящем учебном пособии комплекс программ был реализован с использованием суперкомпьютера СКИФ МГУ "Чебышев".

Проблема собственных значений является одной из наиболее часто встречающихся численных задач. В случае матриц больших размеров требуются параллельные алгоритмы, позволяющие использовать многопроцессорные вычислительные системы.

В настоящем пособии рассматривается решение линейной проблемы собственных значений для симметричных и эрмитовых матриц.

Математическая запись такой задачи для симметричных матриц имеет вид (символом T обозначена операция транспонирования)

$$\mathrm{A}\,\mathrm{z} = \lambda\,\mathrm{z}, \qquad \mathrm{A} = \mathrm{A}^{\mathrm{T}},$$

где

A — симметричная матрица порядка n,

λ — собственные значения матрицы A,

z — собственные векторы матрицы A.

Требуется найти собственные значения λ и соответствующие собственные векторы z, удовлетворяющие указанному выше уравнению.

Математическая запись для эрмитовых матриц имеет вид

$$\mathrm{A}\,\mathrm{z} = \lambda\,\mathrm{z}, \qquad \mathrm{A} = \mathrm{A}^{\mathrm{H}},$$

где

A — эрмитова матрица порядка n,

λ — собственные значения матрицы A,

z — собственные векторы матрицы A,

A^{H} — сопряженная матрица.

Требуется найти собственные значения λ и соответствующие собственные векторы z, удовлетворяющие уравнению выше.

В обоих случаях собственные значения λ являются вещественными.

Одним из наиболее широко известных пакетов программ для решения таких задач является пакет ScaLAPACK (Scalable Linear Algebra PACKage) , позволяющий решать задачи с матрицами больших размеров на распределенной памяти с использованием технологии MPI (Massage Passing Interface) [1, 2].

В НИВЦ МГУ создан русскоязычный аналог этого пакета, получивший название "Комплекс параллельных программ PARALG". В его основе лежат те же принципы реализации параллельных алгоритмов, которые приняты в ScaLAPACKe, и при реализации целевых программ комплекса PARALG использовались базовые модули пакета ScaLAPACK.

В комплексе PARALG содержатся и задокументированы (т.е. снабжены инструкциями по использованию) именно готовые целевые программы решения задач линейной алгебры (в том числе для решения линейной проблемы собственных значений). Комплекс содержит большее количество целевых программ, чем в пакете ScaLAPACK; полный список их приводится в п. 3.7 настоящего пособия. Базовые же модули (как составные части целевых программ) вызываются непосредственно из библиотеки ScaLAPACKa.

Общие правила и принципы, приводимые в теоретической части предлагаемого учебного пособия (п. 2), в одинаковой мере справедливы как для пакета ScaLAPACK, так и для комплекса PARALG. Использование готовых целевых программ комплекса PARALG значительно облегчает эффективное решение указанных задач.

Тем не менее, у неспециалистов в данной области (как показывает опыт) это может вызвать определенные трудности, так как требует знакомства с недостаточно известными действиями и понятиями пакета ScaLAPACK, связанными с обеспечением эффективности параллельных вычислений. Сюда относятся такие, например, особенности параллельной реализации алгоритмов как:

— организация параллельных процессов в так называемую (виртуальную) решетку процессов и сопоставление с выбранной решеткой определенного численного значения параметра, который однозначно характеризует выбранную решетку и называется указателем контекста или просто контекстом;

— специальное распределение блоков матриц по параллельным процессам решетки;

— понятие и правильное определение так называемых дескрипторов матриц

и связанное с этим представление о локальных и глобальных параметрах целевых программ комплекса PARALG.

Кроме того, требуется овладеть правильным использованием подпрограмм пакета BLACS (Basic Linear Algebra Communication Subprograms) [19, 20], входящего в состав пакета ScaLAPACK и выполняющего на более высоком уровне функции примитивов MPI. Указанные особенности обсуждаются в п. 2 данного учебного пособия. Кроме того, в п. 2 приводится краткое описание математических аспектов решения линейной проблемы собственных значений для симметричных и эрмитовых матриц.

В п. 3 разбираются примеры программирования различных случаев при решении линейной проблемы собственных значений для симметричных и эрмитовых матриц с помощью программ комплекса PARALG. Кроме того, описываются

— правила обработки ошибок и выдачи диагностических сообщений;

— запуск соответствующих задач в ОС Linux на суперкомпьютере СКИФ МГУ "Чебышев".

2. Основные правила решения линейной проблемы собственных значений для симметричных и эрмитовых матриц с помощью комплекса PARALG

2.1. Общее описание организации и структуры комплекса

Комплекс программ PARALG относится к разряду "параллельных предметных библиотек", при обращении к которым пользователю не приходится явно применять какие-либо конструкции специальных языков параллельного программирования или интерфейсов, поддерживающих взаимодействие параллельных процессов. Все необходимые действия по совместной работе параллельных процессов при решении задач линейной алгебры выполняются подпрограммами из пакетов PBLAS (Parallel Basic Linear Algebra Subprogramms) [17, 18] и BLACS (Basic Linear Algebra Communication Subprogramms), являющихся составными частями пакета ScaLAPACK.

Пакет BLACS является набором подпрограмм, предназначенных для использования методов распараллеливания при решении задач линейной алгебры. Этот пакет разработан для компьютеров с распределенной памятью и обеспечивает запуск параллельных процессов и их взаимодействие посредством обмена сообщениями. При этом пакт BLACS освобождает пользователей от изучения предназначенных для этих целей комплексов стандартных примитивов, включенных в MPI.

Работа подпрограмм пакета BLACS опирается на представление о так называемой “решетке процессов” (process grid).

Предполагается, что параллельные процессы организованы в двумерную (или одномерную) решетку процессов, в которой каждый из процессов идентифицируется своими координатами в решетке и хранит определенные части обрабатываемых матриц и векторов. Для работы с параллельными процессами в состав пакета BLACS включены подпрограммы, реализующие следующие основные функции:

- построение, изменение и получение сведений о решетке процессов;
- обмен сообщениями (передача матриц или их частей) между двумя процессами;
- передача сообщений от одного процесса сразу многим процессам;
- выполнение необходимых итоговых действий после получения параллельными процессами частичных результатов (например, вычисление итоговых сумм, максимумов или минимумов по всем процессам).

Другим пакетом, к которому обращаются программы комплекса PARALG, является пакет PBLAS, содержащий версии для распределенной памяти подпрограмм (первого, второго и третьего уровней) пакета BLAS (Basic Linear Algebra Subprogram) [22, 23]. В этом пакете содержатся версии подпрограмм, реализующих базовые операции линейной алгебры (например, действия над векторами и скалярами, умножение матрицы на вектор или перемножение двух матриц и др.).

Разработчики пакета PBLAS реализовали его таким образом, чтобы интерфейс его подпрограмм был максимально похожим на интерфейс подпрограмм пакета BLAS.

Таким образом, целевые программы описываемого комплекса PARALG (т.е. программы, которые имеют самостоятельное значение при решении прикладных задач) могут содержать обращения к подпрограммам указанных выше пакетов BLACS, PBLAS и BLAS, а также к базовым и вспомогательным подпрограммам пакета ScaLAPACK.

К базовым подпрограммам пакета ScaLAPACK относятся подпрограммы, которые не используются независимо от других программ комплекса PARALG. Они всегда выступают в роли вызываемых из других подпрограмм. Например, подпрограмма умножения матрицы общего вида на ортогональную матрицу, полученную в результате QR-разложения матрицы другой подпрограммой.

К вспомогательным подпрограммам относятся подпрограммы, называемые

tool routines. Эти подпрограммы выполняют различные вспомогательные функции, например

- выдача параметров машинной арифметики конкретной ЭВМ;
- выдача диагностических сообщений в стандартной форме;
- подсчет количества строк или столбцов распределенной по параллельным процессам матрицы, расположенных на одном из этих процессов, и др.

Целевые программы комплекса PARALG решают задачи из следующих разделов линейной алгебры: решение систем линейных алгебраических уравнений, решение проблемы собственных значений (линейной и обобщенной) и сингулярное разложение матриц. Настоящее пособие предназначено для обучения практическим навыкам использования программ комплекса для решения линейной проблемы собственных значений для симметричных и эрмитовых матриц.

2.2. Применение пакета BLACS для организации параллельных процессов.

Подпрограммы пакета BLACS разработаны специально для обеспечения эффективного решения на параллельных компьютерах с распределенной памятью задач линейной алгебры, в которых основным обрабатываемым объектом является матрица.

При распределении таких двумерных массивов по параллельным процессам удобнее мысленно организовать параллельные процессы в виде логической двумерной решетки процессов. Тогда, например, все части строки распределенной матрицы могут быть найдены в процессах, относящихся к строке решетки процессов, а части столбца — в процессах, относящихся к столбцу решетки процессов.

В принятой в BLACS логической структуре параллельных процессов каждый процесс идентифицируется с помощью пары чисел — координат процесса в решетке процессов.

На рисунке ниже изображена решетка из 8 процессов, состоящая из двух строк и четырех столбцов (2×4).

	0	1	2	3
0				
1				

Этим процессам ставятся в соответствие координаты решетки:
$(0,0)$, $(0,1)$, $(0,2)$, $(0,3)$, $(1,0)$, $(1,1)$, $(1,2)$, $(1,3)$

В общем случае, если решетка процессов имеет R строк и C столбцов, то любой процесс в решетке обозначается парой (i, j), где $0 \leqslant i <$ R, $0 \leqslant j <$ C. Общее число процессов равно $\mathrm{R} \times \mathrm{C}$.

В компьютерах с распределенной памятью каждый параллельный процесс работает со своей локальной памятью, а обмен информацией между процессами осуществляется посредством передачи сообщений через связывающую их сеть.

Пакет BLACS, предназначенный для таких компьютеров, опирается на использование широко известных и распространенных интерфейсов обмена сообщениями MPI (Massage Passing Interface) или PVM (Parallel Virtual Machine). Они содержат не только операции обмена между одним посылающим и одним принимающим сообщение процессом, но также и массовые операции обмена, когда, например, один процесс может рассылать сообщение сразу всем другим параллельным процессам, и, наоборот, один из процессов может получать сообщения от всех других.

В пакете BLACS такие операции называются контекстными операциями (scope operations), а про процессы, принимающие в них участие, говорят, что они находятся внутри операционного контекста (operation's scope). Организация процессов в виде логической решетки, позволяет пакету BLACS естественным образом расширить виды контекстов, а именно, объявлять (задавать) в качестве контекста, т.е. области действия массовых операций обмена сообщениями, не только все параллельные процессы, но также и все процессы, принадлежащие только некоторой строке решетки процессов или некоторому ее столбцу. Это обеспечивает программисту более удобные и гибкие возможности распараллеливания.

Использование понятия решеток процессов не только удобно для программирования, но и позволяет повысить эффективность (скорость) вычислительного процесса. Однако такое возможно только в том случае, если в базовой машине параллельные процессы связаны между собой с помощью, по крайней мере, двумерных сетей.

В большинстве современных суперкомпьютеров это условие выполнено, но при использовании сети Ethernet такое преимущество недостижимо.

2.3. Задание параметров при обращении к целевым программам комплекса.

2.3.1. Глобальные и локальные объекты и параметры параллельных программ.

Программы комплекса PARALG составлены на языке ФОРТРАН-77 в стиле SMPD (Single Program Multiple Data). Это означает, что одна и та же исполняемая программа загружается во все параллельные процессы, заказанные

для решения задачи в команде запуска на счет. В то же время, каждый процесс занимается обработкой своих данных, составляющих некоторую часть общих данных задачи. Эти данные отдельного процесса (их называют локальными данными) непосредственно недоступны другим процессам и хранятся в отдельной (локальной) памяти каждого процесса. Необходимый обмен данными между разными процессами производится только посредством передачи сообщений (технология MPI [1, 2, 10–13]).

Из сказанного следует, что исходную матрицу A не требуется хранить целиком в оперативной памяти процессора как единый массив. На практике такие матрицы (векторы) в виде единого целого могут храниться во внешних файлах (на дисковой памяти). По этой причине такую матрицу A называют **глобальной** матрицей.

Поскольку перед началом работы целевых параллельных программ Комплекса исходные глобальные матрицы (или векторы) разделяются на части и распределяются по параллельным процессам, то в дальнейшем нередко употребляется термин "распределенная глобальная матрица (вектор)". Часть глобальной матрицы (вектора), представленная в локальной памяти одного из параллельных процессов, называется **локальной** частью матрицы (вектора). Совокупность всех локальных частей на всех, используемых при решении задачи параллельных процессах, и составляет **распределенную глобальную** матрицу (вектор).

Поскольку каждый процесс работает со своей локальной частью данных (в нашем случае фрагментом глобальной матрицы), то в головной фортранной программе достаточно выделить (заказать) память, необходимую только для максимальной локальной части этой матрицы из всех локальных частей, обрабатываемых на параллельных процессах.

Локальные части, обрабатываемые на каждом из процессов, могут быть как одинакового, так и разного размера (см. подробнее пп. 2.4, 3.2). Для того чтобы каждый из процессов мог разместить свою локальную часть данных (матрицы), заказываемая область локальной памяти должна быть не меньше максимальной локальной части данных.

Таким образом, в головной фортранной программе, обращающейся к целевой программе комплекса, не требуется резервировать память для всего глобального объекта (матрицы или вектора) целиком, а только для их локальных частей.

Целиком эти объекты существуют, как правило, во внешних файлах. Для сохранения информации о конкретном способе разбиения глобального объекта на блоки и размещении блоков в локальной памяти всех процессов вводится объект, называемый **дескриптором** глобального массива. Дескриптор пред-

ставляет собой одномерный массив, состоящий из 9 или 7 элементов (в зависимости от типа дескриптора). В дескрипторе хранится информация о размерах глобального массива (число строк и столбцов матрицы), о размерах блоков, на которые она разбивается, о номере процесса, в память которого распределяется первый блок глобальной матрицы (левый верхний угол), а также некоторая другая информация. Подробнее о дескрипторах см. в п. 2.3.2.

Учитывая общие для комплекса правила распределения блоков матриц (векторов) по параллельным процессам, по информации в дескрипторах всегда можно определить, в локальной памяти какого процесса и где именно находится элемент глобальной матрицы с глобальными индексами i и j; см. подробнее в п. 2.4.1.

Непосредственное размещение исходных данных в локальной памяти каждого процесса должен выполнять сам пользователь (в головной программе перед обращением к целевой программе комплекса) в соответствии с установленными правилами для матриц разных видов: плотных общего вида или симметричных, ленточных или трехдиагональных.

Организация решетки процессов, обмен информацией между процессами, а также другие действия, связанные с процессами, выполняются с помощью подпрограмм пакета BLACS. Пользователю, однако, в большинстве случаев не требуется знание всех возможностей этого пакета. При решении своей задачи он должен сделать в головной программе лишь несколько стандартных обращений к нескольким подпрограммам пакета BLACS.

Наглядные примеры этого можно найти, например, в предлагаемом пользователям описании целевой параллельной подпрограммы комплекса (в разделе “Пример использования”, см. Приложение 1), а также в фортранном тексте тестового примера к этой подпрограмме (см. Приложение 2). Более подробно описание решеток процессов и правил работы с ними пользователей см. в пп. 2.3.2, 2.4, 3.1.

Из сказанного выше следует, что формальные параметры целевых программ комплекса могут быть глобальными или локальными.

Глобальными называются параметры подпрограмм, которые относятся к указанным выше глобальным объектам. Например, глобальными являются число строк или столбцов исходной матрицы A(M, N) или число наддиагоналей (BWU) в матрице A, если она является ленточной.

Локальными называются параметры, которые характеризуют объекты, определяемые в локальной памяти отдельного процесса. Например, указатель на локальную память, занимаемую локальной частью глобальной матрицы A или глобального вектора B правых частей линейной системы уравнений.

Упоминавшиеся выше дескрипторы массивов характеризуются одновре-

менно и как глобальные, и как локальные параметры подпрограмм, поскольку среди их элементов, определяющих в основном характеристики глобальных массивов, есть характеристика, описывающая размещение части глобального массива в локальной памяти процесса (ведущая размерность локального массива).

2.3.2. Дескрипторы глобальных массивов.

Как уже упоминалось выше, для каждого глобального массива (матрицы или вектора), который необходимо разбить на блоки (с последующим размещением каждого из них в локальной памяти параллельных процессов, участвующих в решении задачи), вводится специальный объект, называемый **дескриптором** массива. Дескриптор представляет собой одномерный массив, состоящий из 9 элементов целого типа (в смысле языка ФОРТРАН). Он предназначен для хранения информации, конкретизирующей способ разбиения глобального массива на блоки и их распределение по всем параллельным процессам.

Дескрипторы являются обязательными параметрами целевых программ комплекса.

Дескрипторы принято обозначать идентификатором DESC с добавлением в конце имени массива (матрицы или вектора). Например, DESCA означает дескриптор матрицы A.

Дескрипторы бывают трех типов, которые обозначаются целыми числами: 1, 501, 502.

Для плотных глобальных матриц A, которые распределяются по двумерной решетке процессов, используется дескриптор, имеющий тип 1 и состоящий из 9 элементов, за каждым из которых закреплено свое символическое имя (см. таблицу ниже).

В таблице указываются номер (индекс) каждого элемента в массиве, его символическое имя (которое оканчивается символом подчеркивания “_”), а также его смысл. Идентификатор, стоящий в символическом имени элемента после символа подчеркивания “_”, является идентификатором глобального массива, описываемого данным дескриптором. Например, N_A обозначает элемент дескриптора, который содержит число столбцов глобальной матрицы A.

Таблица элементов дескриптора плотной глобальной матрицы A

N	Имя		Смысл
1.	DTYPE_A	—	тип дескриптора (для плотной матрицы DTYPE_A=1)
2.	CTXT_A	—	обозначение контекста BLACS'а, соответствующего выбранной решетке процессов, по которой распределяется глобальная матрица A; целое значение, устанавливаемое подпрограммой из пакета BLACS; см. п. 3.1.
3.	M_A	—	число строк в глобальной матрице A
4.	N_A	—	число столбцов в глобальной матрице A
5.	MB_A	—	число строк в блоках, на которые разбивается глобальная матрица A
6.	NB_A	—	число столбцов в блоках, на которые разбивается глобальная матрица A
7.	RSRC_A	—	номер строки процесса в решетке процессов, куда был распределен первый элемент глобальной матрицы A (левый верхний угол); как правило, удобнее всего распределять первый элемент в процесс с номером $(0,0)$
8.	CSRC_A	—	номер столбца процесса в решетке процессов, куда был распределен первый элемент глобальной матрицы A (левый верхний угол); как правило, удобнее всего распределять первый элемент в процесс с номером $(0,0)$
9.	LLD_A	—	ведущая размерность локального массива (LLD_A$\geqslant$MAX$(1,$LOCr(M_A))

В этой таблице и далее число строк локального массива обозначается LOCr(M_A).

Заметим, что записи DESCA(3) и DESCA(M_A) означают один и тот же элемент дескриптора матрицы A, содержащий число строк этой матрицы.

2.4. Алгоритмы распределения матриц по решетке процессов.

2.4.1. Блочно-циклическое отображение плотных матриц в локальную память параллельных процессов.

Как уже говорилось выше, перед обращением программам комплекса PARALG пользователю необходимо разделить исходную матрицу на блоки и распределить ее по параллельным процессам. Для матриц разных видов приняты свои способы разбиения и распределения матриц. Здесь мы рассмотрим наиболее общий случай прямоугольной плотной матрицы общего вида.

Для равномерной загрузки параллельных процессов и эффективного использования подпрограмм пакета BLAS при выполнении операций над плотными глобальными матрицами были разработаны специальные способы разбиения таких матриц на прямоугольные блоки (MB $\times$ NB) и распределения этих блоков

по двумерной решетке параллельных процессов. Простой пример конкретного разбиения плотной матрицы на блоки и распределения их по двумерной решетке процессов, называемого блочно-циклическим отображением, приводится в п. 2.4.2.

Здесь же мы рассмотрим, каким образом блоки, отображенные на один и тот же процесс, располагаются и хранятся в локальной памяти этого процесса. Другими словами, мы выпишем точные формулы, которые связывают элемент глобальной матрицы, определяемый глобальными индексами I и J (т.е. A(I, J)), с координатами процесса, владеющего этим элементом, в решетке процессов (Pr, Pc) и с расположением этого элемента матрицы в локальной памяти этого процесса.

Рассмотрим сначала, для простоты, эти связи на примере одномерного случая, т.е. размещение одномерного глобального массива длины N, разбитого на блоки длины NB, по одномерной решетке из P процессов, перенумерованных, начиная с 0 до (P-1). Элементы самого глобального массива нумеруются от 1 до N. Прежде всего, массив делится на смежные блоки размера NB. Когда N не делится на NB нацело, последний блок массива элементов будет содержать только mod (N, NB) элементов вместо NB элементов. Блоки, на которые разбивается исходный массив, нумеруются (как и процессы), начиная с 0. Они распределяются по процессам подобно тому, как сдается колода карт участникам игры — по кругу (т.е. циклически).

Другими словами, мы предполагаем, что процесс с номером 0 получает первый блок (с номером 0), k-й блок связывается с процессом, имеющим координату (номер) mod (k, P). Блоки, связанные с одним и тем же процессом, хранятся в памяти рядом (в смежных, прилегающих частях). Отображение элемента массива с глобальным индексом I определяется следующим соотношением:

$$I = k * NB + x = (m * P + p) * NB + x,$$

где

- I — глобальный индекс элемента глобального массива;
- m — локальная координата блока, в котором этот элемент расположен;
- p — координата процесса, владеющего этим блоком;
- x — локальная координата элемента внутри того блока, где находится элемент глобального массива с индексом I.

Легко установить соотношения между этими переменными:

$$\begin{aligned} \mathrm{p} &= \big[(\mathrm{I}-1)/\mathrm{NB}\big] \bmod \mathrm{P}, \\ \mathrm{m} &= \big[(\mathrm{I}-1)/(\mathrm{P} * \mathrm{NB})\big], \\ \mathrm{x} &= \bmod(\mathrm{I}-1,\, \mathrm{NB}) + 1. \end{aligned}$$

Эти уравнения позволяют определить локальную информацию (т.е. локальный индекс m * NB+x), так же как и координату процесса p, соответствующую глобальному элементу, идентифицируемому его глобальным индексом I, и наоборот.

В таблице ниже показано отображение в локальную память при разбиении массива на блоки, когда P=2, N=16 и NB=8.

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
m * NB+x	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8

Вовсе не обязательно всегда делать распределение блоков начиная с процесса с номером 0. Иногда, бывает полезно начать распределение данных с процесса, имеющего отличную от нуля координату SRC. В этом случае соотношения становятся такими:

$$\begin{aligned} \mathrm{p} &= \big(\mathrm{SRC} + [(\mathrm{I}-1)/\mathrm{NB}]\big) \bmod \mathrm{P}, \\ \mathrm{m} &= \big[(\mathrm{I}-1)/(\mathrm{P} * \mathrm{NB})\big], \\ \mathrm{x} &= \bmod(\mathrm{I}-1, \mathrm{NB}) + 1. \end{aligned}$$

Ниже в таблице показано размещение блоков при P=2, SRC=1, NB=3 и N=16.

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0
m	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2
x	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1
m * NB+x	1	2	3	1	2	3	4	5	6	4	5	6	7	8	9	7

В двумерном случае предположим, что матрица разделена на $\mathrm{MB} \times \mathrm{NB}$ блоки и что первый блок передан процессу с координатами $(\mathrm{RSRC}, \mathrm{CSRC})$. Формула, приведенная выше для одномерного случая, должна использоваться

повторно независимо для каждого измерения решетки процессов $\mathrm{Pr} \times \mathrm{Pc}$. Например, элемент матрицы (I, J) находится в процессе с координатами $(\mathrm{Pr}, \mathrm{Pc})$ внутри локального блока (m, n) в позиции (x, y), задаваемой формулами

$$
\begin{aligned}
(\mathrm{m}, \mathrm{n}) &= \Big(\big[(\mathrm{I}-1)/(\mathrm{Pr} * \mathrm{MB}) \big], \quad \big[(\mathrm{J}-1)/(\mathrm{Pc} * \mathrm{NB}) \big] \Big), \\
(\mathrm{Pr}, \mathrm{Pc}) &= \Big(\big(\mathrm{RSRC} + [(\mathrm{I}-1)/\mathrm{MB}] \big) \bmod \mathrm{Pr}, \quad \big(\mathrm{CSRC} + [(\mathrm{J}-1)/\mathrm{NB}] \big) \bmod \mathrm{Pc} \Big), \\
(\mathrm{x}, \mathrm{y}) &= \big(\bmod (\mathrm{I}-1, \mathrm{MB}) + 1, \quad \bmod (\mathrm{J}-1, \mathrm{NB}) + 1 \big).
\end{aligned}
$$

Эти формулы показывают, как матрица A размера $\mathrm{M\_A} \times \mathrm{N\_A}$ отображается и хранится на решетке процессов. Матрица сначала разбивается на $\mathrm{MB\_A} \times \mathrm{NB\_A}$ блоки начиная с ее верхнего левого угла. Эти блоки затем равномерно распределяются по решетке процессов циклическим образом.

Каждый процесс владеет набором блоков, которые хранятся рядом (смежно) по столбцам в двумерном массиве, расположенном в памяти по "столбцам".

Это соглашение о локальном размещении позволяет эффективно использовать иерархию локальной памяти посредством вызова пакета BLAS для подмассивов, которые могут быть больше, чем один блок размера $\mathrm{MB\_A} \times \mathrm{NB\_A}$.

На рисунке ниже представлено отображение матрицы 5×5, разделенной на блоки размером 2×2, на решетку процессов размером 2×2 (т.е. M_A=N_A=5, Pr=Pc=2 и MB_A=NB_A=2). Локальные элементы каждого столбца матрицы хранятся рядом в памяти каждого процесса.

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

	0			1	
0	a_{11}	a_{12}	a_{15}	a_{13}	a_{14}
	a_{21}	a_{22}	a_{25}	a_{23}	a_{24}
	a_{51}	a_{52}	a_{55}	a_{53}	a_{54}
1	a_{31}	a_{32}	a_{35}	a_{33}	a_{34}
	a_{41}	a_{42}	a_{45}	a_{43}	a_{44}

Цифры 0 и 1 в левой и верхней части последней таблицы означают номера строк и столбцов решетки процессов соответственно. Важной задачей для

пользователя является определение числа строк и столбцов плотной глобальной матрицы, которое получает каждый конкретный процесс. Для выполнения этой функции существует служебная подпрограмма NUMROC.

Для локального числа строк используется обозначение LOCr ( ), а для локального числа столбцов — LOCc ( ). Значения LOCr ( ) и LOCc ( ), получаемые подпрограммой NUMROC, являются результатом точных вычислений.

Однако если требуется понять общую идею вычисления размера локального массива, то можно проделать следующее грубое вычисление верхней границы этой величины:

а) верхняя граница для LOCr ( ) оценивается по формуле

$$\mathrm{LOCr}\,(\ ) = \frac{\dfrac{\mathrm{M\_A} + \mathrm{MB\_A} - 1}{\mathrm{MB\_A}} + \mathrm{Pr} - 1}{\mathrm{Pr}} * \mathrm{MB\_A}$$

или

$$\mathrm{LOCr}\,(\ ) = \big[[\mathrm{M\_A}/\mathrm{MB\_A}]/\mathrm{Pr}\big] * \mathrm{MB\_A};$$

б) величина LOCc ( ) оценивается по формуле

$$\mathrm{LOCc}\,(\ ) = \frac{\dfrac{\mathrm{N\_A} + \mathrm{NB\_A} - 1}{\mathrm{NB\_A}} + \mathrm{Pc} - 1}{\mathrm{Pc}} * \mathrm{NB\_A}$$

или

$$\mathrm{LOCc}\,(\ ) = \big[[\mathrm{N\_A}/\mathrm{NB\_A}]/\mathrm{Pc}\big] * \mathrm{NB\_A}.$$

Заметим, что эти вычисления могут привести к очень большой переоценке величины действительно требующегося пространства.

2.4.2. Пример блочно-циклического распределения плотной матрицы по решетке процессов.

Приводимый в настоящем разделе пример является простой иллюстрацией блочно-циклического распределения плотной глобальной матрицы A по двумерной решетке процессов.

В соответствии с принятой схемой, сначала плотная матрица A размера $\mathrm{M} \times \mathrm{N}$ разделяется на блоки размера $\mathrm{MB} \times \mathrm{NB}$ начиная с левого верхнего угла этой матрицы. Эти блоки затем равномерно распределяются по каждому измерению решетки процессов. Математические соотношения, соответствующие такой схеме распределения, приводятся в п. 2.4.1.

Таким образом каждый процесс владеет набором блоков, которые расположены в его локальной памяти рядом в двумерном массиве, хранящемся по столбцам (см. ниже).

Ниже на рисунке показано разделение матрицы A размером 9×9 на блоки размером 2×2.

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

Далее показано отображение полученных блоков на решетку процессов размером 2×3 (двумерное блочно-циклическое распределение данных).

Чтобы понять, как распределятся изображенные выше блоки по процессам решетки, сначала напишем на каждой из клеток (блоков) рисунка координаты процессов, куда они должны быть распределены в соответствии с установленным блочно–циклическим распределением, описанным в п. 2.4.1.

(0,0)	(0,1)	(0,2)	(0,0)	(0,1)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)
(0,0)	(0,1)	(0,2)	(0,0)	(0,1)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)
(0,0)	(0,1)	(0,2)	(0,0)	(0,1)

Пусть первый блок распределяется в процесс $(0, 0)$. Тогда все блоки, расположенные с ним в той же строке будут распределяться в ту же строку решетки процессов, т.е. первая координата в этих клетках будет равна 0.

Вторая же координата (столбца решетки) будет циклически изменяться: $0, 1, 2, 0, 1$, поскольку столбцов в решетке только 3 (с координатами $0, 1, 2$).

Все блоки, расположенные с первым блоком в том же самом столбце, будут распределяться в тот же самый столбец решетки процессов. Иными словами вторая координата в этих клетках будет равна 0. Первая же координата (строки решетки) будет циклически изменяться: $0, 1, 0, 1, 0$, поскольку число строк в решетке только 2 (с координатами 0 и 1).

Другими словами, каждая координата независимо от другой (в строках — слева направо, а в столбцах — сверху вниз), циклически изменяется.

Если теперь мы соединим вместе (пристыкуем) все блоки, имеющие одинаковые координаты процесса, то получим массив элементов, который и должен расположиться в локальной памяти процесса с такими координатами.

Сборка клеток (блоков) с одинаковыми координатами делается так:

- из этих клеток выбирается расположенная левее и выше всех;
- к ней пристыковываются снизу клетки, расположенные в этом же столбце, тем самым получается первый столбец блоков;
- затем берется клетка, расположенная правее в той же строке клеток, что и самая первая; к ней пристыковываются все расположенные ниже в том же столбце, тем самым получаем следующий столбец клеток;
- после того, как собраны все столбцы клеток с одинаковыми координатами, пристыковываем их по-очереди справа к первому столбцу с теми же координатами.

Тем самым получаем единый двумерный массив, расположенный в локальной памяти процесса с указанными в клетках координатами.

Ниже представлено расположение элементов исходной матрицы во всех процессах решетки после завершения процесса блочно–циклического распределения. Вверху указаны номера столбцов решетки процессов, слева — номера строк решетки процессов.

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

Ниже в таблице указаны характеристики локальных массивов для каждого из процессов решетки.

Координаты процесса	LLD_A	LOCr (M_A)	LOCc (N_A)
(0,0)	5	5	4
(0,1)	5	5	3
(0,2)	5	5	2
(1,0)	4	4	4
(1,1)	4	4	3
(1,2)	4	4	2

Число строк LOCr и число столбцов LOCc матрицы A, которыми владеет конкретный процесс, могут отличаться у разных процессов в решетке. Подобно этому, для каждого процесса в решетке процессов существует ведущая локальная размерность LLD. Ее величина может быть различной для разных процессов в решетке процессов. Например, как мы можем видеть на рисунке выше, локальный массив, хранящийся в строке решетки процессов с номером 0, должен иметь ведущую локальную размерность LLD не меньше 5, а хранящийся в строке с номером 1 — не меньше 4.

2.5. Алгоритмы решения линейной проблемы собственных значений для симметричных и эрмитовых матриц.

2.5.1. Пусть A является вещественной симметричной матрицей порядка n.

Проблема собственных значений для симметричной матрицы A порядка n состоит в нахождении собственных значений λ и соответствующих собственных векторов z, удовлетворяющих уравнению

$$Az = \lambda z, \quad A = A^T. \tag{1}$$

Символом T обозначена операция транспонирования. Собственные значения λ являются вещественными.

Сначала вещественная симметричная матрица A ортогональным преобразованием подобия [8, 9] на основе метода отражений приводится к виду

$$A = QTQ^T, \tag{2}$$

где

Q — ортогональная матрица,

T — вещественная симметричная трехдиагональная матрица.

Собственные значения матрицы T вычисляются на основе QR-алгоритма (или QL-алгоритма) [8, 9].

Поскольку матрицы A и T ортогонально подобны, их собственные значения совпадают.

Собственные векторы матрицы T вычисляются методом обратных итераций [8, 9] .

Собственные векторы матрицы A вычисляются по формуле

$$Z = QS,$$

где S — матрица, столбцы которой образуют собственные векторы матрицы T.

2.5.2. Пусть A является эрмитовой матрицей порядка n.

Проблема собственных значений для эрмитовой матрицы A порядка n состоит в нахождении собственных значений λ и соответствующих собственных векторов z, удовлетворяющих уравнению

$$Az = \lambda z, \quad A = A^H. \tag{3}$$

Символом H обозначена операция комплексного сопряжения. Собственные значения λ являются вещественными.

Сначала эрмитова матрица A ортогональным преобразованием подобия на основе метода отражений приводится к виду

$$A = UTU^H, \tag{4}$$

где

U — унитарная матрица,

T — вещественная симметричная трехдиагональная матрица.

Собственные значения матрицы T вычисляются на основе QR-алгоритма (или QL-алгоритма).

Поскольку матрицы A и T ортогонально подобны, их собственные значения совпадают.

Собственные векторы матрицы T вычисляются методом обратных итераций.

Собственные векторы матрицы A вычисляются по формуле

$$Z = US,$$

где S — матрица, столбцы которой образуют собственные векторы матрицы T.

3. Практические сведения по использованию параллельных программ.

3.1. Предварительные действия, необходимые при обращении к программам.

Рассмотрим общие правила программирования при использовании целевых программ Комплекса.

С учетом сказанного в п. 2.2, прежде всего необходимо начинать с определения конкретной решетки процессов. В программах комплекса PARALG используются следующие обозначения:

NPROW — число строк решетки процессов,

NPCOL — число столбцов решетки процессов,

NP=NPROW*NPCOL — общее число процессов решетки.

Для идентификации каждого процесса используется пара (MYROW, MYCOL):

MYROW — номер строки процесса в решетке,

MYCOL — номер столбца процесса в решетке.

При этом

$$0 \leqslant \text{MYROW} < \text{NPROW},$$
$$0 \leqslant \text{MYCOL} < \text{NPCOL}.$$

Еще одна процедура, которую необходимо проделать пользователю перед обращением к одной из целевых программ комплекса, это разделить исходную(ые) матрицу(ы) на блоки и распределить эти блоки по определенным правилам по всем процессам решетки.

Это необходимо проделать по той причине, что программы ориентированы на более эффективную блочную реализацию операций с матрицами. Поэтому при обращении к подпрограммам комплекса необходимо задавать, в частности, величину указанных блоков:

MB — число строк матрицы в блоке;

NB — число столбцов матрицы в блоке.

Описание конкретных правил распределения матриц по параллельным процессам приводится далее в п. 3.2, а также в п. 2.4.

Проделав указанные действия, пользователь может вызвать одну из программ комплекса, передав ей через параметры всю информацию об исходных матрицах, а также о том, как распределены их элементы в локальную память каждого из процессов решетки.

После выполнения требуемых вычислений и получения результатов пользователю необходимо освободить использованную решетку процессов и осуществить выход из пакета BLACS.

Ниже подробнее описываются общие обязательные действия программиста при обращении к любой целевой программе Комплекса.

3.1.1. Инициализация решетки процессов.

После присвоения конкретных значений параметрам решетки NPROW и NPCOL инициализация решетки процессов проще всего производится посредством обращения к подпрограмме SL_INIT из разряда служебных (tool routines). Она состоит из обращений к необходимым подпрограммам пакета BLACS.

Эта подпрограмма инициализирует выбранную решетку процессов, упорядочивая процессы по строкам, а также присваивает служебному параметру ICTXT некоторое целое значение. Это значение закрепляет за выбранной пользователем решеткой процессов статус конкретной области действия массовых

операций передачи сообщений. Параметр ICTXT называется указателем контекста, системным контекстом, контекстом BLACS'а или просто контекстом.

Пользователь, работая с выбранной решеткой, не должен производить никаких действий с параметром ICTXT. Необходимо только передавать его в качестве входного параметра при обращении к другим подпрограммам пакета BLACS, а также к подпрограммам комплекса, куда передаются сведения об обрабатываемых матрицах.

Обращение к подпрограмме SL_INIT выглядит так:

CALL SL_INIT (ICTXT, NPROW, NPCOL),

где

ICTXT — специфицирует контекст BLACS'а (глобальный выходной параметр, тип: целый);

NPROW — число строк, в создаваемой решетке процессов (глобальный входной параметр, тип: целый);

NPCOL — число столбцов, в создаваемой решетке процессов (глобальный входной параметр, тип: целый).

Программы комплекса написаны с расчетом на использование в стиле SPMD (Single Program Multiple Data). Это означает, что одна и та же программа пользователя будет загружена и одновременно выполняться всеми процессами решетки. Поэтому, при необходимости выполнения некоторых действий только на некоторых из этих процессов, требуется определить координаты процесса, на котором выполняется данный экземпляр программы. Как уже указывалось, они обычно обозначаются идентификаторами (MYROW, MYCOL). Например, пусть необходимо произвести распечатку входных параметров, одинаковых для всех процессов. Чаще всего это делают с процесса с координатами (0, 0).

Для определения координат процесса используют подпрограмму BLACS_GRIDINFO пакета BLACS:

CALL BLACS_GRIDINFO (ICTXT, NPROW, NPCOL, MYROW, MYCOL),

где

MYROW — номер строки данного процесса в решетке процессов, $0 \leqslant \text{MYROW} < \text{NPROW}$ (глобальный выходной параметр);

MYCOL — номер столбца данного процесса в решетке процессов, $0 \leqslant \text{MYCOL} < \text{NPCOL}$ (глобальный выходной параметр).

Три первых параметра описаны выше, только данном случае параметр ICTXT — входной.

3.1.2. Распределение матриц по решетке процессов.

Поскольку каждый процесс производит операции над своей частью исходной глобальной матрицы, до обращения к программам комплекса необходимо распределить по определенным правилам матрицу A по решетке процессов. Это является обязанностью пользователя (подробности этого процесса описаны в п. 2.4, а также в п 3.2).

Каждой глобальной матрице, распределенной по решетке процессов, должен быть приписан дескриптор, содержащий информацию о том, как именно было произведено это распределение. Дескриптор представляет собой одномерный массив из 9 или 7 элементов целого типа. Подробное описание дескрипторов можно найти в п. 2.3.2. Здесь мы покажем, как инициализировать дескриптор глобальной матрицы A (DESCA).

Если матрица A — плотная, то проще всего инициализировать дескриптор посредством обращения к служебной подпрограмме DESCINIT.

Ниже приводится обращение к этой подпрограмме при инициализации дескриптора DESCA. Подобные обращения можно найти, например, в текстах тестовых примеров к программам TDSYEV1.f и TZHEEV1.f вычисления всех собственных значений вещественной симметричной или эрмитовой матриц.

CALL DESCINIT(DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA, INFO).

Этот вызов подпрограммы DESCINIT эквивалентен следующим операторам присваивания:

```
DESCA(1) = 1
DESCA(2) = ICTXT
DESCA(3) = M
DESCA(4) = N
DESCA(5) = MB
DESCA(6) = NB
DESCA(7) = RSRC
DESCA(8) = CSRC
DESCA(9) = MXLLDA
```

Здесь первый элемент обозначает тип дескриптора для плотной матрицы, второй — обсуждавшийся выше служебный параметр подпрограмм пакета BLACS (контекст).

Другие параметры имеют следующий смысл:

M, N — число строк и столбцов глобальной матрицы A(M,N);

MB, NB — число строк и столбцов в блоках, на которые разбивается матрица A;

RSRC,CSRC — координаты процесса в решетке, куда размещается пользователем первый элемент распределяемой глобальной матрицы;

MXLLDA — ведущая локальная размерность матрицы A;

INFO — целая переменная, служащая для сообщения о результате работы программы (выходной параметр).

Простой пример отображения глобальной матрицы на решетку процессов можно найти, например, в тестовом примере к подпрограмме вычисления собственных значений вещественной симметричной матрицы PDSYEV1.f (см. TDSYEV1.f). В нем эти действия выполняются в подпрограмме с именем MATINIT (или PDMATINIT), которая является частью тестового примера.

3.1.3. Освобождение решетки процессов.

После того как необходимые вычисления на решетке процессов были завершены, требуется освободить решетку процессов посредством вызова подпрограммы
BLACS_GRIDEXIT. Когда завершены все вычисления, выход из пакета BLACS должен быть осуществлен посредством обращения к подпрограмме BLACS_EXIT.

Типичный пример фрагмента программы, включающий в себя эти действия:

```
CALL BLACS_GRIDEXIT(ICTXT)
CALL BLACS_EXIT(0).
```

3.2. Программирование распределения матриц по решетке процессов.

В настоящем учебном пособии рассматриваются три способа реализации на языке ФОРТРАН рассылки элементов исходных глобальных матриц по параллельным процессам, поскольку каждый из них удобнее использовать в трех разных ситуациях.

1. Первый способ удобен только для небольших матриц и требует от пользователя самостоятельного размещения элементов исходных матриц в локальной памяти параллельных процессов в соответствии с общими правилами, описанными в п. 2.4. При этом используются обычные операторы присваивания языка ФОРТРАН. Такой способ достаточно трудоемок и может использоваться для целей обучения и усвоения общих правил распределения матриц (см. п. 2.4).

Фортранная подпрограмма такого способа распределения приводится, например, в конце Приложения 1.

2. Второй способ удобнее как для небольших матриц, так и для матриц среднего размера с регулярной структурой или матриц любого размера, элементы которых задаются математической формулой.

Этот способ гораздо проще для пользователя, поскольку не требует самостоятельного определения местоположения элементов матриц в локальной памяти каждого из параллельных процессов. Он опирается на использование служебной подпрограммы PDELSET, которая сама (в соответствии с упомянутыми общими правилами, см. п. 2.4.1) заносит элемент с индексами I и J исходной матрицы A(I,J) в определенное место локальной памяти того или другого из параллельных процессов.

На ФОРТРАНе это выливается в написание некоторого числа циклов, внутри которых стоит обращение к подпрограмме PDELSET.

Фортранная подпрограмма с реализацией такого способа распределения приводится в конце Приложения 2.

Еще раз подчеркнем, что хотя в конце Приложения 1 и Приложения 2 приводятся два разных способа распределения одной и той же исходной матрицы, результат распределения будет одним и тем же. Подробный разбор приводимого в этих приложениях примера дается в п. 3.4.

3. Третий способ может быть использован для обработки больших исходных матриц, численное значение элементов которых хранится в виде файлов на внешней памяти (наиболее частый на практике случай).

Тогда можно воспользоваться специальной служебной подпрограммой, последовательно считывающей из внешнего файла элементы исходной матрицы и распределяющей их по установленным правилам в локальную память того или другого из параллельных процессов.

Пример обращения к такой служебной фортранной подпрограмме PDLAREAD приводится в Приложении 3. Однако пользователь может написать свою аналогичную подпрограмму, если какие-либо детали размещения исходной матрицы в файле отличаются от приводимого примера.

В этом же примере содержится обращение к служебной подпрограмме PDLAWRITE записи полученных результатов в файл. Тексты этих подпрограмм можно найти, перейдя по гиперссылкам в конце Интернет-страницы с адресом

http://num-anal.srcc.msu.ru/par_prog/instr.htm.

3.3. О документировании и примерах использования параллельных программ.

3.3.1. Правила наименований подпрограмм.

Каждая подпрограмма комплекса PARALG имеет имя, начинающееся с буквы P. Допущено нарушение стандарта языка Фортран, так как имена подпрограмм могут быть длиной более шести символов; кроме того, в именах служебных подпрограмм (tool routines) допускается использование символа подчеркивания " _ ".

Все целевые и базовые подпрограммы имеют имена в виде последовательности шести или семи символов: PXYYZZ или PXYYZZZ, где второй символ X может принимать следующие значения, указывающие на тип обрабатываемых данных:

S	REAL
D	DOUBLE PRECISION
C	COMPLEX
Z	DOUBLE COMPLEX

Следующие две буквы YY указывают на то, какого вида матрица обрабатывается подпрограммой (или вид самой главной из участвующих матриц). Большинство из этих двухсимвольных сочетаний относятся одновременно как к вещественным, так и к комплексным матрицам, некоторые же относятся конкретно к одному из этих типов матриц. Приняты следующие мнемонические правила для указания вида обрабатываемых матриц:

DB	(general band)	—	ленточная общего вида, для которой не требуется выбор ведущего элемента
DT	(general tridiagonal)	—	трехдиагональная общего вида, для которой не требуется выбор ведущего элемента
GB	(general band)	—	ленточная общего вида
GE	(general)	—	общего вида (т.е. не симметричная, иногда прямоугольная)
GG	(general matrices, generalized problem)	—	матрица общего вида, обобщенная проблема собственных значений (т.е. пара матриц общего вида)
HE	(complex Hermitian)	—	комплексная эрмитова
OR	(real orthogonal)	—	вещественная ортогональная
PB	(symmetric or Hermitian positive definite band)	—	симметричная или эрмитова ленточная положительно определенная
PO	(symmetric or Hermitian positive definite)	—	симметричная или эрмитова положительно определенная

PT	(symmetric or Hermitian positive definite tridiagonal)	—	симметричная или эрмитова трехдиагональная положительно определенная
ST	(real symmetric tridiagonal)	—	вещественная симметричная трехдиагональная
SY	(symmetric)	—	симметричная
TR	(triangular or in some cases quasi-triangular)	—	трехдиагональная или в некоторых случаях почти трехдиагональная
TZ	(trapezoidal)	—	трапециевидная
UN	(complex unitary)	—	комплексная унитарная

Последние два символа ZZ или три ZZZ символа в имени подпрограммы указывают ее назначение. Например, окончание TRF означает подпрограммы, которые выполняют факторизацию матриц.

Имена вспомогательных подпрограмм подчиняются тем же правилам, за исключением того, что третьим и четвертым символом YY обычно являются символы LA (например, PDLASCL или PZLARFG).

3.3.2. Описания программ и примеры их использования.

Документация параллельных программ по линейной алгебре включает в себя

- информацию общего характера (об организации и структуре Комплекса и об общих правилах использования программ и размещения данных);
- систематический каталог целевых программ;
- описания целевых программ (т.е. инструкции по их использованию).

Документация общего характера приводится как в рамках настоящего учебного пособия, так и в соответствующих подразделах раздела “Комплекс программ по линейной алгебре для вычислений на распределенной памяти” Научно-образовательного Интернет-ресурса НИВЦ МГУ по численному анализу (http://num-anal.srcc.msu.ru/). Там же приводится систематический каталог целевых программ в виде таблиц, содержащих, в частности, гипер-ссылки на описания программ.

Описания программ составлялись во многом по аналогии с тем, как это было ранее разработано и реализовано в Библиотеке фортранных программ по численному анализу НИВЦ МГУ [3], которая состоит из обычных последовательных алгоритмов.

Такая четкая форма документации, состоящая из нескольких обязательных смысловых разделов, заслужила широкое одобрение среди многочисленных

пользователей Библиотеки численного анализа, в течение многих лет эксплуатировавших ее в разных организациях и на разных вычислительных системах.

Одним из самых важных разделов в описании каждой целевой программы является его последний раздел под названием “Пример использования”. Содержание этого раздела на примере матриц небольшого размера наглядно демонстрирует пользователю, как будут распределены их элементы по параллельным процессам и как такое распределение можно реализовать средствами языка ФОРТРАН и служебных подпрограмм. А именно, приводится фрагмент головной фортранной программы (теста) с подпрограммой рассылки элементов конкретной матрицы по параллельным процессам одним из трех способов, рассматриваемых в настоящем учебном пособии (см. п. 3.2). Указанные подпрограммы рассылки имеют в примерах либо имя MATINIT, либо PDMATINIT.

3.3.3. Обозначения в примерах по использованию программ комплекса.

В тестовых примерах и примерах по использованию программ комплекса приняты следующие единообразные обозначения величин, задающих параметры исходной задачи и выбранный способ распараллеливания (указаны в алфавитном порядке).

Имя		**Определение**
CSRC	—	номер столбца решетки процессов, в который распределяется первый столбец исходной глобальной матрицы
DESCA	—	дескриптор исходной глобальной матрицы A
DESCB	—	дескриптор исходной глобальной матрицы B (или правых частей при решении систем линейных уравнений, или при решении обобщенной проблемы собственных значений)
ICTXT	—	контекст BLACS’a, связанный с выбранной решеткой процессов (внутренний параметр комплекса; служит для обеспечения согласованной работы подпрограмм комплекса и устанавливается пользователем обращением к соответствующей подпрограмме BLACS’a в начале решения задачи, см. пп. 3.1, 2.3.2)
M	—	число строк исходной глобальной матрицы A
MB	—	число строк в блоке, на которые делится исходная матрица A

MXLLDA — Максимальная ведущая локальная размерность локальной части матрицы A среди всех локальных частей на всех процессах решетки

MXLLDB — максимальная ведущая локальная размерность локальной части матрицы B среди всех локальных частей на всех процессах решетки

MXLOCC — максимальное число столбцов в локальных частях матрицы A, которые принадлежат процессам в столбцах решетки процессов

MXLOCR — максимальное число строк в локальных частях матрицы A, которые принадлежат процессам в строках решетки процессов

MXRHSC — максимальное число столбцов в локальных частях матрицы (вектора) правых частей B, которые принадлежат процессам в столбцах решетки процессов (если B — вектор, то MXRHSC=1)

MYCOL — координата (номер) столбца вызываемого процесса в решетке процессов

MYROW — координата (номер) строки вызываемого процесса в решетке процессов

N — число столбцов исходной глобальной матрицы A (и число строк глобальной матрицы (вектора) правых частей B при решении систем уравнений)

NB — число столбцов в блоке, на которые делится исходная матрица A

NBRHS — число столбцов в блоке, на которые делится глобальная матрица (вектор) правых частей системы уравнений B (если B — вектор, то NBRHS=1)

NPCOL — число столбцов в решетке процессов

NPROW — число строк в решетке процессов

NRHS — число столбцов исходной глобальной матрицы (вектора) правых частей системы уравнений B (если B — вектор, то NBRHS=1)

RSRC — Номер строки решетки процессов, в которую распределяется первая строка исходной глобальной матрицы

3.3.4. Упрощения, принятые в примерах по использованию программ комплекса.

Опишем далее некоторые упрощения и ограничения, которые были сделаны при подготовке примеров использования подпрограмм комплекса, для того чтобы упростить пользователю освоение правил работы с этими подпрограммами.

1. В примерах предполагается, что RSRC=CSRC=0; это означает, что исходные матрицы A (и B) распределяются по решетке процессов начиная с процесса $(0, 0)$. На самом деле, любой из процессов решетки может быть выбран в качестве начального при распределении матриц.

2. В примерах предполагается, что задача решается при использовании всей исходной распределенной матрицы A (хотя подпрограммы допускают, чтобы задачи решались с матрицей, являющейся подматрицей sub (A) исходной матрицы A). При указанном предположении параметры, задающие первый элемент подматрицы A, полагаются равными 1, т.е. IA=JA=IB=JB=1.

3. При распределении исходных матриц A по решетке процессов используется подпрограмма MATINIT (см. конец Приложения 1), которая заносит с помощью операторов присваивания в соответствующие локальные массивы на каждом из процессов решетки распределенные на него элементы исходных матриц. Упрощает эту процедуру использование служебной подпрограммы PDELSET (см. п. 3.2 и конец Приложения 2).

4. В примерах предполагается, что ведущая локальная размерность локальных частей матрицы A является одинаковой для всех процессов, принадлежащих к одним и тем же строкам решетки процессов (т.е. имеющих одинаковые координаты строк решетки). Переменная MXLLDA равна максимальной ведущей локальной размерности для матрицы A (обозначаемой LLD_A) из всех строк решетки процессов. Однако в общем случае локальные ведущие размерности локальных частей матриц могут отличаться друг от друга для всех процессов решетки.

5. В примерах часто рассматриваются исходные матрицы небольших размеров. При этом размеры блоков, на которые разбиваются эти матрицы, выбираются небольшими. Например, MB=NB=2. Однако это не является самым лучшим размером блоков для практических задач. Для достижения лучшей производительности более подходящим размером блоков будет MB=NB=32 или MB=NB=64.

3.4. Пример использования программы для симметричных матриц.

Рассмотрим подробнее пример по использованию подпрограммы PDSYEV2, которая предназначена для вычисления всех собственных значений и собственных векторов вещественной симметричной матрицы общего вида A. Данный пример приводится в описании этой подпрограммы в разделе “Пример исполь-

зования" (см. Приложение 1).

Пример демонстрирует решение проблемы собственных значений для симметричной матрицы A размера 7×7. Ниже приводится ее изображение в символическом и числовом виде.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

Порядок матрицы N=7.

Сначала мы должны задать решетку процессов и распределить по ней исходную глобальную матрицу A.

Как говорилось в пп. 3.1 и 2.4.1, базовые подпрограммы Комплекса реализуют специальные блочные алгоритмы обработки матриц, позволяющие повысить скорость решения задачи. При этом распределение плотных матриц по параллельных процессам происходит блочно-циклическим образом (см. пп. 2.4.1, 2.4.2 и п. 3.2).

Пусть матрица разбивается на блоки размером MB×NB, где MB=NB=2.

1	0	0	0	0	0	1
0	1	0	0	0	0	2
0	0	1	0	0	0	3
0	0	0	1	0	0	4
0	0	0	0	1	0	5
0	0	0	0	0	1	6
1	2	3	4	5	6	7

Пусть пропуск программы осуществляется на 4 процессах, которые образуют решетку 2×2, т.е. NPROW=2 и NPCOL=2 (см. пп. 2.2, 3.1).

После распределения полученных блоков по процессам решетки (по принятым правилам, которые описаны в пп. 2.4.1, 2.4.2), имеем следующие рисунки распределения элементов матрицы A по процессам (в символическом и числовом виде), на которых верху указаны номера столбцов решетки процессов, слева — номера строк решетки процессов.

	0				1		
0	a_{11}	a_{12}	a_{15}	a_{16}	a_{13}	a_{14}	a_{17}
	a_{21}	a_{22}	a_{25}	a_{26}	a_{23}	a_{24}	a_{27}
	a_{51}	a_{52}	a_{55}	a_{56}	a_{53}	a_{54}	a_{57}
	a_{61}	a_{62}	a_{65}	a_{66}	a_{63}	a_{64}	a_{67}
1	a_{31}	a_{32}	a_{35}	a_{36}	a_{33}	a_{34}	a_{37}
	a_{41}	a_{42}	a_{45}	a_{46}	a_{43}	a_{44}	a_{47}
	a_{71}	a_{72}	a_{75}	a_{76}	a_{73}	a_{74}	a_{77}

	0				1		
0	1	0	0	0	0	0	1
	0	1	0	0	0	0	2
	0	0	1	0	0	0	5
	0	0	0	1	0	0	6
1	0	0	0	0	1	0	3
	0	0	0	0	0	1	4
	1	2	5	6	3	4	7

Из последнего рисунка видно, что процесс с координатами $(0,0)$ содержит локальный массив (часть матрицы A) размера $(4,4)$, процесс $(0,1)$ — локальный массив размера $(4,3)$, процесс $(1,0)$ — локальный массив размера $(3,4)$, процесс $(1,1)$ — локальный массив размера $(3,3)$.

В Приложении 1 (раздел "Пример использования") приведен способ инициализации элементов распределенной матрицы A с помощью обычных операторов присваивания (см. подпрограмму MATINIT), каждый из которых выполняется только на одном из параллельных процессов.

Такую подпрограмму пользователь может написать, только заранее представив себе распределение элементов матрицы A в соответствии с последним изображенным рисунком, а также учитывая, что в языке Фортран двумерные массивы хранятся в памяти по столбцам.

Такой способ инициализации распределенной матрицы A явно не годится для матриц больших размеров. Поэтому в Приложении 2 приводится более простой способ инициализации матрицы A с помощью обращения к служебной подпрограмме PDELSET, которая сама определяет (в соответствии с принятыми правилами блочно-циклического распределения), на какой из параллельных процессов и в какое место локальной памяти (для локальной части матрицы A) следует поставить конкретный элемент глобальной матрицы A с глобальными индексами I иJ т.е. A(I,J).

Оба приведенных в двух Приложениях способа распределения матрицы A приводят к одному и тому же результату.

Еще один способ распределения исходной матрицы по параллельным процессам (возможно, имеющий наибольшее практическое применение) приводится в Приложении 3. Это случай считывания элементов исходной матрицы из внешнего файла. Такое считывание и последующее распределение ее элементов по решетке параллельных процессов выполняется с помощью служебной подпрограммы PDLAREAD.

Этот способ распределения матрицы является наиболее автоматизированным. Он требует от пользователя только правильного выделения локальной памяти под локальные части исходной матрицы, массива собственных векторов и рабочего массива.

Вычислить размеры локальной части распределенного двумерного массива можно с помощью служебной подпрограммы NUMROC, описание функций которой приводится в п. 2.3.2.

Приводимая в Приложении 3 головная программа демонстрирует также запись полученных результатов (т.е. собственных значений и собственных векторов) во внешний файл. Собственные векторы, части которых располагаются в локальной памяти параллельных процессов, собираются и записываются одним глобальным массивом в выходной файл, содержимое которого также приведено в конце Приложения 3.

Полный текст описанного в Приложении 3 примера использования подпрограммы PDSYEV2 можно получить также в системе Интернет по следующему адресу в разделе “Систематический каталог”:

http://num-anal.srcc.msu.ru/par_prog/

3.5. Пример использования программы для эрмитовых матриц.

Рассмотрим подробнее пример по использованию подпрограммы PZHEEV4, которая предназначена для вычисления собственных значений, принадлежащих заданному интервалу, и соответствующих собственных векторов эрмитовой матрицы A.

Рассматривается решение проблемы собственных значений для эрмитовой матрицы A размера 7×7.

Для простоты демонстрация делается на примере матрицы, аналогичной рассмотренной в предыдущем пункте (п. 3.4), т.е. подпрограмме PZHEEV4 подается частный случай эрмитовой матрицы, все мнимые части элементов которой равны 0.

Ниже приводится ее изображение в числовом виде.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1+i*0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 2+i*0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 3+i*0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 4+i*0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 5+i*0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 6+i*0 \\ 1-i*0 & 2-i*0 & 3-i*0 & 4-i*0 & 5-i*0 & 6-i*0 & 7 \end{pmatrix}$$

Порядок матрицы N=7.

Пусть интервал, в котором ищутся собственные значения (VL, VU) равен $(2, 20)$.

Сначала зададим решетку процессов и распределим по ней исходную глобальную матрицу A.

Пусть матрица разбивается на блоки размером MB×NB, где MB=NB=2.

1	0	0	0	0	0	$1+i*0$
0	1	0	0	0	0	$2+i*0$
0	0	1	0	0	0	$3+i*0$
0	0	0	1	0	0	$4+i*0$
0	0	0	0	1	0	$5+i*0$
0	0	0	0	0	1	$6+i*0$
$1-i*0$	$2-i*0$	$3-i*0$	$4-i*0$	$5-i*0$	$6-i*0$	7

Пусть пропуск программы осуществляется на 6 процессах, которые образуют решетку 3×2, т.е. NPROW=3 и NPCOL=2 (см. пп. 2.2, 3.1).

После распределения полученных блоков по процессам решетки (по принятым правилам, которые описаны в п. 2.4.1), имеем следующую картину распределения элементов матрицы A по процессам (в числовом виде; вверху указаны номера столбцов решетки процессов, слева — номера строк решетки процессов.)

	0			1		2	
0	1	0	$1+i*0$	0	0	0	0
	0	1	$2+i*0$	0	0	0	0
	0	0	$5+i*0$	0	0	1	0
	0	0	$6+i*0$	0	0	0	1
1	0	0	$3+i*0$	1	0	0	0
	0	0	$4+i*0$	0	1	0	0
	$1-i*0$	$2-i*0$	7	$3-i*0$	$4-i*0$	$5-i*0$	$6-i*0$

Из последнего рисунка видно, что процесс с координатами $(0,0)$ содержит локальный массив (часть матрицы A) размера $(3,4)$, процесс $(0,1)$ — локальный массив размера $(2,4)$, процесс $(0,2)$ — локальный массив размера $(2,4)$, процесс $(1,0)$ — локальный массив размера $(3,3)$, процесс $(1,1)$ — локальный массив размера $(2,3)$, процесс $(1,2)$ — локальный массив размера $(2,3)$.

Приведем здесь фортранные операторы второго, из рассмотренных в п. 3.2, более простого способа реализации распределения на ФОРТРАНе исходной матрицы в локальную память каждого из параллельных процессов с помощью служебной подпрограммы PZELSET (предназначенной для комплексных матриц).

```
*
      DO 2 J = 1, N
      DO 1 I = 1, N
      CALL PZELSET( A, I, J, DESCA, (0.0D0, 0.0D0) )
   1 CONTINUE
   2 CONTINUE
      DO 4 J = 1, N
      DO 3 I = 1, N
         IF( I.EQ.J ) THEN
            CALL PZELSET( A, I, J, DESCA, 1.0D0 )
         END IF
   3 CONTINUE
   4 CONTINUE
      J = N
      DO 5 I = 1, N
      CALL PZELSET( A, I, J, DESCA, DCMPLX( DBLE(I), 0.0D0) )
   5 CONTINUE
      I = N
      DO 6 J = 1, N
      CALL PZELSET( A, I, J, DESCA, DCMPLX( DBLE(J), 0.0D0) )
   6 CONTINUE
Полностью весь фортранный текст рассматриваемого примера приведен
в tzheev46.zip.
В результате получено одно собственное значение и соответствующий собственный
вектор.

Собственное значение:

  W(1) = 14.0D0
```

```
Собственный вектор:

 Z( 1, 1 ) =  0.620173672946042268D-01 + i * 0.0D+0
 Z( 2, 1 ) =  0.124034734589208454D+00 + i * 0.0D+0
 Z( 3, 1 ) =  0.186052101883812687D+00 + i * 0.0D+0
 Z( 4, 1 ) =  0.248069469178416907D+00 + i * 0.0D+0
 Z( 5, 1 ) =  0.310086836473021155D+00 + i * 0.0D+0
 Z( 6, 1 ) =  0.372104203767625430D+00 + i * 0.0D+0
 Z( 7, 1 ) =  0.806225774829855024D+00 + i * 0.0D+0
```

Вычислить размеры локальной части распределенного двумерного массива можно с помощью служебной подпрограммы NUMROC, описание которой приводится в п. 2.3.2.

3.6. Обработка ошибок и выдача диагностических сообщений.

Ошибочные ситуации, возникающие в процессе выполнения программ, вызывающих программы комплекса, можно разделить на три уровня.

3.6.1. Первый уровень представляют ситуации, диагностируемые самими как целевыми, так и базовыми программами комплекса, в частности, подпрограммами используемого комплексом пакета PBLAS.

Во всех целевых и базовых подпрограммах комплекса имеется параметр INFO, располагаемый последним в списке параметров. Параметр INFO предназначен для выдачи диагностики о результатах работы подпрограммы.

Общее правило состоит в том, что в случае успешного завершения работы подпрограммы параметру INFO присваивается значение 0, в случае обнаружения каких-либо ошибок при проверке входных фактических параметров — отрицательное целое значение, а в случае обнаружения в процессе счета ситуации, приводящей к невозможности получения необходимого результата, — положительное целое значение.

Более подробно в случае обнаружения ошибки в фактическом параметре параметру INFO присваивается:

- номер ошибочного параметра в списке параметров подпрограммы со знаком минус, если он представляет собой скалярное значение;
- номер ошибочного параметра в списке, умноженный на 100 (если этот параметр — массив), плюс номер ошибочного элемента в этом массиве, вся эта сумма также берется со знаком минус.

Последний случай используется в основном для диагностирования ошибок

в элементах дескрипторных массивов (или дескрипторов), см. “Дескрипторы глобальных массивов”.

При диагностировании ошибки во входном параметре вызывается служебная подпрограмма обработки ошибок PXERBLA ( ), которая выдает сообщение, содержащее имя подпрограммы, обнаружившей ошибку, и номер ошибочного параметра, например

“On entry to PDGESV parameter number 4 had an illegal value”.

Обычная версия подпрограммы PXERBLA ( ), сделав выдачу диагностического сообщения, не останавливает выполнение программы. Это делается из-за того, что некоторые “ошибки” могут быть исправимыми, и пользователю предоставляется возможность продолжить выполнение программы. В случае если это не устраивает пользователя, он может использовать свой вариант PXERBLA ( ), добавив туда вызов подпрограммы BLACS_ABORT ( ) из пакета BLACS [19, 20], которая прекратит выполнение программы пользователя.

В том случае, если ошибка во входном параметре была обнаружена подпрограммой комплекса достаточно высокого уровня (т.е. целевой или базовой), пользователь имеет возможность исправить такую ошибку и продолжить выполнение программы, так как после выдачи диагностического сообщения выполняется оператор RETURN.

Если же ошибка обнаружена в подпрограмме низкого уровня, то она считается неисправимой и подпрограмма PXERBLA ( ) выдает сообщение и выполнение прерывается посредством вызова BLACS_ABORT ( ).

Все целевые и базовые подпрограммы Комплекса выполняют проверку как локальных, так и глобальных входных параметров. Вспомогательные же подпрограммы, как правило, не выполняют проверку входных параметров.

Подпрограммы пакета PBLAS в целях эффективности выполняют проверку правильности только локальных параметров из своего списка параметров. Если ошибка обнаруживается хотя бы на одном из параллельных процессов текущего контекста, то выполнение программы прерывается.

Проверка правильности глобальных параметров, передаваемых в подпрограммы
PBLAS’а, выполняется в вызывающих процедурах более высокого уровня.

Отсутствие глобальной проверки в подпрограммах PBLAS’а является следствием высокой стоимости (в смысле ухудшения производительности) таких проверок.

Если параметр INFO при выходе из подпрограммы комплекса имеет значение больше нуля то, как упоминалось в начале п. 3.6.1, это означает, что нежелательная ситуация возникла уже после проверки входных параметров,

т.е. в процессе счета. Это бывает по следующим основным причинам:

- матрица оказывается вырожденной (в пределах рабочей точности);
- симметричная матрица не является положительно определенной, как это необходимо;
- итерационный алгоритм для вычисления собственных значений или собственных векторов не сходится за установленное количество итераций.

Например, если используется подпрограмма PSGESV2 для решения системы уравнений с вещественной матрицей одинарной точности, которая является близкой к вырожденной, может быть зафиксирована вырожденность в пределах рабочей точности на i-й итерации LU-факторизации.

В этом случае либо параметру INFO присваивается значение i, либо (что более вероятно) может быть вычислена оценка обратного числа обусловленности, которая окажется меньше, чем относительная машинная точность; в этом случае INFO полагается равным $(n + 1)$.

Если возникла ситуация, когда выдается INFO>0, то управление всегда возвращается в вызывающую подпрограмму, а подпрограмма PXERBLA ( ) не вызывается и сообщение об ошибке не выдается. Поэтому при выходе из подпрограммы Комплекса всегда следует проверять значение параметра INFO.

Ошибка с INFO>0 может получиться, например, в одном из следующих случаев.

- Была использована неподходящая подпрограмма. Например, когда подпрограмма заканчивается неудачно из-за того, что симметричная матрица не является положительно определенной. Тогда следует воспользоваться другой подпрограммой, предназначенной для знаконеопределенных (indefinite) матриц.
- Была использована подпрограмма для одинарной точности, когда необходимо было использовать подпрограмму для двойной точности. Например, если подпрограмма PSGESV2 диагностирует матрицу, близкую к вырожденной (как указывалось выше), то соответствующая подпрограмма PDGESV2 может оказаться в состоянии решить эту задачу (только если она не является плохо обусловленной).
- Ошибочная ситуация может возникнуть, если для теоретически хорошо обусловленной и положительно определенной матрицы делается ошибка в процессе генерирования ее элементов, передаваемых подпрограмме (т.е. нарушается одно из указанных свойств матрицы).

- Была сделана программистская ошибка (особенно неопытным программистом), например, такого рода как:

  задание неверного числа параметров при обращении к подпрограмме,

  неверный порядок параметров,

  неверный тип фактического параметра,

  неверная размерность массива,

  недостаточная величина рабочего пространства,

  присвоение неверного значения входному параметру.

Специализированные реализации могут вызывать специфические для конкретных систем средства обработки исключительных ситуаций либо с помощью вспомогательных подпрограмм обработки ошибок, либо прямо из самих подпрограмм.

Кроме того, можно использовать специальные программы тестирования, передающие набор ошибочных параметров в подпрограмму, которые вызывают те или иные исключительные ситуации, и проверяют правильность их обнаружения.

3.6.2. Ко второму уровню ошибочных ситуаций, возникающих при выполнении подпрограмм комплекса, можно отнести ситуации, которые могут быть обработаны подпрограммами пакета BLACS. Реакция BLACS'а на ошибочные ситуации зависит от того, какой отладочный уровень был установлен во время компиляции программы.

Этот уровень устанавливается посредством использования макроса препроцессора языка Си BlacsDebugLvl.

Чтобы узнать, какой уровень отладки используется в BLACS'е в данном случае, можно использовать подпрограмму BLACS_GET ( ).

Если установлен уровень 0, то выполняется небольшое количество проверок ошибок (т.е. несколько критических ситуаций). Например, подпрограмма BLACS_GRIDINIT ( ) не позволит пользователю создать решетку процессов с большим количеством процессов, чем имеется в наличии.

Однако пакет BLACS большинство параметров не проверяет, поскольку это приведет к ухудшению производительности.

В этой связи пользователям рекомендуется при связывании своей программы с библиотекой BLACS'а устанавливать при компиляции отладочный уровень 1 до тех пор, пока идет процесс отладки программы. При этом проверяется большая часть параметров. Кроме того, выдаются и некоторые другие полезные сообщения. Например, пользователь будет предупрежден, если какой-либо

процесс посылает сообщение сам себе. Такая ситуация допустима, однако свидетельствует о плохом программировании и требует достаточно большого буферного пространства, потому что в этом случае может произойти “зависание” при передаче больших сообщений.

Во многих случаях программа с установленным уровнем отладки 0 будет просто зависать, оставляя разработчика в неведении, что же произошло. Это может быть вызвано, например, попыткой получить сообщение от процесса, который находится вне текущего контекста. При отладочном уровне 1 пакет BLACS может обнаружить такую ошибку пользователя и выдать соответствующее сообщение.

Все сообщения BLACS'а можно разделить на три следующих вида.

Предупреждения. BLACS обнаружил нехорошую ситуацию, которую, однако, можно либо исправить, либо проигнорировать. Выдается предупреждающее сообщение с помощью внутренней подпрограммы BlacsWarn, и выполнение программы продолжается.

Сообщение выдается в виде:

BLACS WARNING 'текст сообщения'

from {<p>,<q>}, pnum = <pnum>, Contxt = <ictxt>, on line <#> of file '<fname>',

здесь

{<p>, <q>}	—	координаты процесса, выдавшего сообщение, в решетке процессов;
<pnum>	—	номер процесса, выдаваемый как выходное значение первого параметра подпрограммой BLACS_PINFO;
<ictxt>	—	целое значение контекста. Обратите внимание на то, что это значение не является одинаковым для всех процессов решетки. Например, процесс $\{0,0\}$ может иметь ictxt=0, а процесс $\{0,1\}$ имеет ictxt=1 для того же самого контекста. Однако, <pnum> и <ictxt> вместе обеспечивают однозначную идентификацию процесс/контекст;
<#>	—	номер строки в файле с именем <fname>, который выдает сообщение;
<fname>	—	имя файла, в котором содержится подпрограмма, выдавшая сообщение.

Ошибки. BLACS обнаружил ошибку. Выдается сообщение об ошибке посредством внутренней подпрограммы BlacsErr, и выполнение программы оста-

навливается посредством вызова подпрограммы BLACS_ABORT ( ). Сообщение при этом выдается в таком же виде, как и в случае предупреждения, за исключением того, что слова BLACS WARNING заменяются на слова BLACS ERROR.

Не вся указанная информация может иметься в наличии в момент выдачи диагностического сообщения. Например, если ошибка выдается до того, как была создана решетка процессов, координаты процесса в решетке будут недоступны. Для любого значения, которое BLACS не может изобразить, выдается значение -1, указывающее, что оно неизвестно.

С помощью этих двух главных, указанных выше, подпрограмм выдачи сообщений об ошибках, программист может относительно легко изменить обработку ошибок, если предлагаемые варианты программ не отвечают его потребностям.

Одной из особенно раздражающих ситуаций является та, что на многих вычислительных системах при выдаче на экран проходит слишком много времени до ее завершения. Подпрограмма BlacsErr может таким образом завершить выполнение, до того как выдача достигнет экрана, и сообщение об ошибке будет потеряно. В этом случае, пользователь может сделать так, чтобы BlacsErr подождала, пока сообщение будет выдано на экран, перед тем как выполнение будет остановлено, либо, например, вообще отказаться от остановки.

Системная ошибка. BLACS получил сообщение об ошибке от операционной системы. Он передает его пользователю и завершает выполнение программы.

3.6.3. К третьему уровню ошибок относятся системные ошибки.

При работе с подпрограммами комплекса иногда могут возникать сообщения об ошибках, передаваемые операционной системой. В предыдущем п. 3.6.2 описана реакция BLACS'а на эту ситуацию.

Эти сообщения будут зависеть от конкретной операционной системы и используемой на ней версии BLACS'а. Для того чтобы понять сообщение, пользователю может потребоваться документация от производителей или просмотр страниц руководства на экране (manpages), описывающих системные сообщения об ошибках.

Например, если используется PVM BLACS, то будет выдан номер ошибки в стиле PVM (Parallel Virtual Machine). Тогда для понимания сообщения об ошибке, т.е. перевода номера сообщения в соответствующий текст, может потребоваться, например, краткое руководство по PVM.

3.7. Общий список программ комплекса для решения линейной алгебраической проблемы собственных значений для симметричных и эрмитовых матриц.

В п. 2.5 настоящего пособия были перечислены алгоритмы, используемые в Комплексе при решении линейной алгебраической проблемы собственных значений для симметричных и эрмитовых матриц.

Приведем здесь список всех целевых программ, включенных в настоящее время в раздел Комплекса “Решение линейной проблемы собственных значений для симметричных и эрмитовых матриц”.

PDSYEV1, PZHEEV1 — вычисление всех собственных значений вещественной симметричной (эрмитовой) матрицы с использованием QL- или QR-алгоритма

PDSYEV2, PZHEEV2 — вычисление всех собственных значений и собственных векторов вещественной симметричной (эрмитовой) матрицы с использованием QL- или QR-алгоритма

PDSYEV3, PZHEEV3 — вычисление собственных значений вещественной симметричной (эрмитовой) матрицы, принадлежащих заданному интервалу с использованием QL- или QR-алгоритма

PDSYEV4, PZHEEV4 — вычисление собственных значений, принадлежащих заданному интервалу, и соответствующих собственных векторов вещественной симметричной (эрмитовой) матрицы с использованием QL- или QR-алгоритма

PDSYEV5, PZHEEV5 — вычисление собственных значений, принадлежащих заданному интервалу индексов, вещественной симметричной (эрмитовой) матрицы с использованием QL- или QR-алгоритма

PDSYEV6, PZHEEV6 — вычисление собственных значений вещественной симметричной (эрмитовой) матрицы, принадлежащих заданному интервалу индексов, и соответствующих собственных векторов с использованием QL- или QR-алгоритма

Дадим здесь некоторые пояснения к реализации указанных в п. 2.5 алгоритмов с использованием базовых подпрограмм комплекса.

Описанное в п. 2.5 разложение (2) вещественной матрицы выполняется базовой подпрограммой PDSYTRD. В комплексном случае разложение (4) из п. 2.5 выполняется базовой подпрограммой PZHETRD. Подпрограммы PDSYTRD и PZHETRD представляют матрицу Q или U как произведение элементарных

матриц отражения.

Напомним, что элементарные матрицы отражения представляются в виде

$$H(i) = I - \tau \times v \times v', \quad 1 \leqslant i \leqslant n,$$

где τ — вещественный скаляр, v — вещественный вектор-столбец и v' — вещественная вектор-строка.

Для вычисления всех собственных значений и собственных векторов матрицы T из формул (2) и (4) (п. 2.5) в целевых подпрограммах PDSYEV1, PZHEEV1, PDSYEV2, PZHEEV2 используются базовые подпрограммы DSTEQR2 и ZSTEQR2 соответственно. Эти подпрограммы являются модификациями версий подпрограмм DSTEQR и ZSTEQR из пакета LAPACK [14]. Они выполняют меньшее число итераций, чем подпрограммы пакета LAPACK благодаря усовершенствованию техники предсказания (the look-ahead technique). Некоторые дополнительные модификации позволяют каждому процессу выполнять частичную корректировку матриц Q или U. Эти подпрограммы вычисляют собственные значения и собственные векторы симметричной трехдиагональной матрицы на основе неявного QL- или QR-алгоритма.

Когда требуется вычислить некоторые собственные значения матрицы T, в целевых подпрограммах PDSYEV3, PZHEEV3, PDSYEV4, PZHEEV4, PDSYEV5, PZHEEV5, PDSYEV6, PZHEEV6 используется базовая подпрограмма PDSTEBZ. Она позволяет вычислять либо некоторые собственные значения, лежащие в заданном интервале, либо собственные значения, принадлежащие интервалу индексов от i-го до j-го в массиве собственных значений, упорядоченных по возрастанию.

Собственные векторы матрицы T вычисляются с помощью базовых подпрограмм PDSTEIN или PZSTEIN (в комплексном случае). При заданных собственных значениях подпрограммы PDSTEIN и PZSTEIN используют обратные итерации для вычисления некоторых или всех собственных векторов.

Приведем ниже список базовых подпрограмм Комплекса, используемых для решения линейной алгебраической проблемы собственных значений для симметричных и эрмитовых матриц.

PDSYNTRD, PZHENTRD	—	приведение симметричной (эрмитовой) матрицы к трехдиагональной форме методом отражений
DSTEQR2, ZSTEQR2	—	вычисление всех собственных значений и собственных векторов симметричной трехдиагональной матрицы

PDSTEBZ — вычисление собственных значений (всех, в заданном интервале или в интервале индексов) симметричной трехдиагональной матрицы

PDSTEIN, PZSTEIN — вычисление собственных векторов симметричной (эрмитовой) трехдиагональной матрицы методом обратных итераций

PDORMTR, PZUNMTR — умножение матрицы общего вида на матрицу отражения, вычисленную подпрограммой PDSYTRD (PZHETRD)

3.8. Запуск программ комплекса в ОС Linux на суперкомпьютере "Чебышев".

Комплекс программ PARLAG доступен пользователям суперкомпьютера СКИФ "Чебышев" в НИВЦ МГУ в виде откомпилированных библиотек. Пакеты (BLAS, BLACS, ScaLAPACK и др.), модули которых используются программами комплекса, входят в состав MKL-библиотеки (Intel Math Kernel Library), установленной на этом суперкомпьютере, и должны быть подсоединены при получении исполнимой программы. Объектные модули самого комплекса берутся из библиотеки libparalg.a, указанной в последней строке приводимого ниже скрипта **bld-tcl**, который предназначен для компиляции и получения исполнимой программы для ее последующего запуска на счет в параллельном режиме на суперкомпьютере "Чебышев".

```
#! /bin/sh
EXEDIR=.
EXE="<имя исполнимой программы>"
OBJ="<список имен объектных модулей>"
F77OPTS="-O2"
F77=mpif77
#
$F77 -c $F77OPTS <имя исходного фортранного модуля>
#
echo Linking executable $EXE
#
$F77 -o $EXEDIR/$EXE $OBJ \
       -Wl,--start-group -lmkl_scalapack_lp64 -lmkl_blacs_lp64 \
       -lmkl_intel_lp64 \
       -lmkl_sequential -lmkl_core \
       -Wl,--end-group \
       -lpthread -libparalg.a
```

Здесь:

- <имя исполнимой программы> — указываемое пользователем имя, которое будет присвоено полученному исполнимому модулю (например, tcl_syev1.e);
- <список имен объектных модулей> — список имен всех объектных модулей, которые не вызываются из библиотек (например, tcl_syev1.o);
- $F77 -c $F77OPTS <имя исходного фортранного модуля> — команда компиляции (которых может быть несколько) фортранного модуля, например: $F77 -c $F77OPTS tcl_syev1.f.

В скрипте **bld-tcl** подключаются все пакеты из MKL-библиотеки, содержащие все вызываемые подпрограммы, используемые при работе параллельных подпрограмм комплекса.

Полученная исполнимая программа может быть затем запущена на счет с помощью команды **mpirun** с параметрами.

Например, если требуется запустить программу tcl_syev1.e на шести процессорах, то команда запуска имеет вид

mpirun -np 6 -maxtime 1 ./tcl_syev1.e

В тестах программ выдача исходных матриц и полученных результатов осуществляется с помощью обращения к подпрограмме PDLAPRNT из пакета ScaLAPACK(TOOLS). Эта подпрограмма распечатывает всю распределенную матрицу (вектор) полностью по столбцам по одному элементу в строке, получая элементы со всех участвующих в вычислениях процессоров.

Обращение к программам комплекса может осуществляться с помощью подпрограмм автоматизации доступа к ним. Описание этих подпрограмм доступно по адресу

http://num-anal.srcc.msu.ru/par_prog/org/avt.htm

Тогда конкретный вид скрипта может быть проиллюстрирован следующим примером для запуска на счет программы PDSYEV1:

```
#! /bin/sh
EXEDIR=.
EXE="tcl_syev1.e"
OBJ="tcl_syev1.o cal_syev1.o pdsyev1.o pdlaprnt.o pdlaread.o pdlawrite.o"
#
F77OPTS="-O2"
F77=mpif77
#
$F77 -c $F77OPTS tcl_syev1.f
```

```
#
$F77 -o $EXEDIR/$EXE $OBJ \
       -Wl,--start-group -lmkl_scalapack_lp64 -lmkl_blacs_lp64 \
       -lmkl_intel_lp64 \
       -lmkl_sequential -lmkl_core \
       -Wl,--end-group \
       -lpthread -libparalg.a
```

Чтобы не указывать все объектные модули подпрограмм комплекса PARALG, которые необходимо вызвать для решения конкретной задачи, достаточно указать в последней строке скрипта объектную библиотеку libparalg.a. Тогда строка с объектными модулями будет содержать только объектный модуль головной программы OBJ="tcl_syev1.o".

Здесь подпрограммы pdlaread.o и pdlawrite.o — входящие в состав комплекса служебные подпрограммы для считывания из файла исходных матриц и распределения их по параллельным процессам и записи в файл полученного вектора решения.

Приложение 1

Описание программы для вычисления всех собственных значений и собственных векторов вещественной симметричной матрицы.

Подпрограмма: PDSYEV2
Назначение
Вычисление всех собственных значений и собственных векторов вещественной симметричной матрицы
Математическое описание

На первом этапе подпрограмма приводит симметричную матрицу A к трехдиагональной форме методом отражений: $A = Q * T * Q^T$, где Q — ортогональная матрица, а T — симметричная трехдиагональная матрица. На втором этапе неявным QL- или QR-алгоритмом находятся собственные значения матрицы T, которые совпадают с собственными значениями матрицы A, поскольку матрицы A и T ортогонально подобны. Собственные значения и собственные векторы вычисляются с машинной точностью.

Литература: http://num_anal.srcc.msu.ru/par_prog/
Использование

CALL PDSYEV2 (UPLO, N, A, IA, JA, DESCA, W, Z, IZ, JZ, DESCZ, WORK, LWORK, INFO)

Параметры

UPLO — если UPLO = 'U', то задается верхний треугольник матрицы A; если UPLO = 'L', то задается нижний треугольник матрицы A (глобальный входной параметр, тип символьный);

N — порядок распределенной подматрицы sub (A); $N \geqslant 0$ (глобальный входной параметр, тип целый);

A — массив двойной точности, распределенный по процессам блочно-циклическим образом (см. п. 2.4), глобальная размерность которого (N, N), а локальная размерность (LLD_A, LOCc (JA + N - 1));
на входе: исходная симметричная матрица A;
если UPLO = 'U', то используется только верхняя треугольная часть матрицы A;
если UPLO = 'L', то используется только нижняя треугольная часть матрицы A;
на выходе: нижний треугольник (при UPLO = 'L') или верхний треугольник (при UPLO = 'U') матрицы A, включая диагональ, не сохраняется (локальный входной параметр, локальное рабочее пространство);

IA — глобальный номер строки матрицы A, который указывает на начало подматрицы (глобальный входной параметр, тип целый);

JA — глобальный номер столбца матрицы A, который указывает на начало подматрицы (глобальный входной параметр, тип целый);

DESCA — дескриптор распределенной матрицы A (одномерный массив длины DLEN);
если тип дескриптора одномерный (DTYPE_A = 501), то DLEN $\geqslant$ 7,
если тип дескриптора двумерный (DTYPE_A = 1), то DLEN $\geqslant$ 9;
DESCA содержит информацию о размещении A в памяти; полное описание DESCA см. в п. 2.3.2 (глобальный и локальный входной параметр, тип целый);

W — одномерный массив двойной точности длины N; если INFO = 0, то собственные значения располагаются в нем по возрастанию (глобальный выходной параметр);

Z – массив двойной точности с глобальной размерностью (N, N) и с локальной размерностью (LLD_Z, LOCc (JZ + N - 1)); содержит ортонормированные собственные векторы симметричной матрицы A (локальный выходной параметр);

IZ — глобальный номер строки матрицы Z, который указывает на начало подматрицы sub (Z) (глобальный входной параметр, тип целый);

JZ — глобальный номер столбца матрицы Z, который указывает на начало подматрицы sub (Z) (глобальный входной параметр, тип целый);

DESCZ — дескриптор распределенной матрицы Z (одномерный массив длины DLEN);
если тип дескриптора одномерный (DTYPE_Z = 501), то DLEN $\geqslant$ 7,
если тип дескриптора двумерный (DTYPE_Z = 1), то DLEN $\geqslant$ 9;
DESCZ содержит информацию о размещении Z в памяти; полное описание DESCZ см. в п. 2.3.2; DESCZ (CTXT_) должен быть равен DESCA (CTXT_)
(глобальный и локальный входной параметр, тип целый);

WORK — одномерный рабочий массив двойной точности длины LWORK;
на выходе в элементе WORK (1) возвращается необходимая длина рабочего пространства
(локальное рабочее пространство, локальный выходной параметр);

LWORK — задаваемая длина рабочего пространства WORK;
минимальная требуемая величина LWORK вычисляется самой подпрограммой, если к ней обратиться со значением LWORK = -1; при этом подпрограмма не производит никаких других вычислений, а требуемое значение LWORK возвращается в элементе массива WORK (1); (локальный или глобальный входной параметр, тип целый);

INFO — целая переменная, диагностирующая результат работы подпрограммы (глобальный выходной параметр)
= 0 — успешное завершение работы;
< 0 — если i-й фактический параметр подпрограммы является массивом и его j-й элемент имеет недопустимое значение, то INFO = -(i ∗ 100 + j); если i-й фактический параметр является скаляром и имеет недопустимое значение, то INFO = - i;
>0 — если INFO = i, где 1 ⩽ i ⩽ N, то i-е собственное значение не может быть вычислено с машинной точностью после выполнения 30 ∗ N итераций;
если INFO = N + 1, то подпрограмма не гарантирует точности вычисленных собственных значений.

Версии

PSSYEV2, PCHEEV2, PZHEEV2 — вычисление всех собственных значений и собственных векторов симметричной (эрмитовой) матрицы для случаев вещественных данных одинарной точности, комплексных данных одинарной точности, комплексных данных двойной точности соответственно

Вызываемые подпрограммы

Здесь указаны только базовые подпрограммы (подпрограммы 2-го уровня), которые вызываются из целевой программы (программа 1-го уровня).

PDSYTRD, PZHETRD — приведение симметричной (эрмитовой) матрицы к трехдиагональной форме методом отражений

DSTEQR2, ZSTEQR2 — вычисление всех собственных значений и собственных векторов симметричной трехдиагональной матрицы

PDORMTR, PZUNMTR — умножение матрицы общего вида на матрицу отражения, вычисленную подпрограммой PDSYTRD (PZHETRD)

PDLAMCH — вычисление машинных параметров для арифметики с плавающей запятой

PDLASET — внедиагональные элементы матрицы полагаются равными ALFA, а диагональные элементы — равными BETA

PDLANSY — вычисляет значения первой нормы, нормы Фробениуса, бесконечной нормы или наибольшего абсолютного значения любого элемента вещественной симметричной, комплексной симметричной или эрмитовой матрицы

PDLASCL — умножает вещественную матрицу на вещественный скаляр

Замечания по использованию

1. В подпрограммах PSSYEV2, PCHEEV2, PZHEEV2 параметры A, WORK и Z имеют тип REAL, COMPLEX и DOUBLE COMPLEX соответственно, а параметр W — тип REAL, REAL и DOUBLE PRECISION соответственно

Список параметров подпрограммы PZHEEV2 имеет следующий вид:

PZHEEV2(UPLO, N, A, IA, JA, DESCA, W, Z, IZ, JZ, DESCZ, WORK, LWORK, RWORK, LRWORK, INFO),

где дополнительные параметры RWORK и LRWORK означают:

RWORK — одномерный рабочий массив двойной точности длины LRWORK; на выходе в элементе RWORK(1) возвращается необходимая длина рабочего массива RWORK (локальное рабочее пространство, локальный выходной параметр);

LRWORK — задаваемая длина рабочего пространства RWORK; LRWORK $\geqslant$ 2 * N + 2 * N - 2 (локальный входной параметр, тип целый).

2. Используются подпрограммы BLACS_GRIDINFO (из пакета BLACS), DCOPY, DSCAL (из пакета BLAS), LSAME, INDXG2P, NUMROC, CHK1MAT, PCHK2MAT,
PXERBLA, SL_GRIDRESHAPE, PDELGET (из библиотеки ScaLAPACK_TOOLS).

Пример использования

Вычисление всех собственных значений и собственных векторов симметричной матрицы A, которая имеет вид

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

Порядок матрицы N = 7.
Пусть матрица разбивается на блоки с размером блоков NB = 2.
Осуществляется пропуск программы на 4 процессах, которые образуют решетку 2 на 2. Подробный разбор данного примера см. п. 3.4.
Фрагмент фортранного текста вызывающей программы. (принятые в тестах обозначения приведены в п. 3.3).
В Приложении 2 приведен другой (более простой) способ инициализации и распределения матрицы A посредством обращения к служебной подпрограмме PDELSET.

```
      PROGRAM TDSYEV21
      include 'mpif.h'
      INTEGER            LWORK, MAXN, LDA, MAXPROCS, NOUT
      DOUBLE PRECISION   ZERO, MONE
      PARAMETER          ( MAXN = 100, LDA = 100, LWORK = 264,
     $                   MAXPROCS = 512, NOUT =6, ZERO = 0.0D+0,
     $                   MONE = -1.0D+0 )
      CHARACTER          UPLO
      PARAMETER          ( UPLO = 'U' )

      INTEGER            CTXT, I, IAM, INFO, MYCOL, MYROW, N, NB,
     $                   NPCOL, NPROCS, NPROW, IA, JA, IZ, JZ

      INTEGER            DESCA( 9 ), DESCZ( 9 )
      DOUBLE PRECISION   A( LDA, LDA ), Z( LDA, LDA ), W( MAXN ),
     $                   WORK( LWORK ), WORK1( 7 )

      EXTERNAL           BLACS_EXIT, BLACS_GET, BLACS_GRIDEXIT,
     $                   BLACS_GRIDINFO, BLACS_GRIDINIT, BLACS_PINFO,
     $                   BLACS_SETUP, DESCINIT, MATINIT, PDLAPRNT,
     $                   PDSYEV2
```

```
      N = 7
      NB = 2
      NPROW = 2
      NPCOL = 2
      IA = 1
      JA = 1
      IZ = 1
      JZ = 1

      CALL BLACS_PINFO( IAM, NPROCS )
      IF( ( NPROCS.LT.1 ) ) THEN
         CALL BLACS_SETUP( IAM, NPROW*NPCOL )
      END IF

      CALL BLACS_GET( -1, 0, CTXT )
      CALL BLACS_GRIDINIT( CTXT, 'R', NPROW, NPCOL )
      CALL BLACS_GRIDINFO( CTXT, NPROW, NPCOL, MYROW, MYCOL )

      IF( MYROW.EQ.-1 ) GO TO 20

      CALL DESCINIT( DESCA, N, N, NB, NB, 0, 0, CTXT, LDA, INFO )
      CALL DESCINIT( DESCZ, N, N, NB, NB, 0, 0, CTXT, LDA, INFO )
*
* Построение матрицы A
*
      CALL MATINIT( N, A, IA, JA, DESCA )
*
*  Вычисление всех собственных значений и собственных векторов
*
      CALL PDSYEV2( UPLO, N, A, IA, JA, DESCA, W,
     $              Z, IZ, JZ, DESCZ, WORK, LWORK, INFO)
*
      CALL BLACS_GRIDEXIT( CTXT )
 20   CONTINUE

      CALL BLACS_EXIT( 0 )
      STOP
      END
*
```

```
      SUBROUTINE MATINIT( N, A, IA, JA, DESCA )
*
* MATINIT генерирует и распределяет матрицу A по решетке процессов
*
      INTEGER           LLD_, CTXT_
      PARAMETER         ( LLD_ = 9, CTXT_ = 2 )
      INTEGER           IA, JA, N
      INTEGER           DESCA( * )
      DOUBLE PRECISION  A( * )
      EXTERNAL          BLACS_GRIDINFO
      INTEGER           MXLLDA, MYCOL, MYROW, NPCOL, NPROW
      DOUBLE PRECISION  C, K
*
* Выполняемые операторы
*
      CALL BLACS_GRIDINFO( DESCA( CTXT_ ), NPROW, NPCOL, MYROW, MYCOL )
*
      C = 0.0D0
      K = 1.0D0
*
      MXLLDA = DESCA( LLD_ )
*
* Локальная часть матрицы A для процесса (0, 0)
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         A( 1 ) = K
         A( 2 ) = C
         A( 3 ) = C
         A( 4 ) = C
         A( 1+MXLLDA ) = C
         A( 2+MXLLDA ) = K
         A( 3+MXLLDA ) = C
         A( 4+MXLLDA ) = C
*
         A( 1+2*MXLLDA ) = C
         A( 2+2*MXLLDA ) = C
         A( 3+2*MXLLDA ) = K
         A( 4+2*MXLLDA ) = C
*
```

```
         A( 1+3*MXLLDA ) = C
         A( 2+3*MXLLDA ) = C
         A( 3+3*MXLLDA ) = C
         A( 4+3*MXLLDA ) = K
*
* Локальная часть матрицы A для процесса (0, 1)
*
      ELSE IF( MYROW.EQ.0 .AND. MYCOL.EQ.1 ) THEN
         A( 1 ) = C
         A( 2 ) = C
         A( 3 ) = C
         A( 4 ) = C
*
         A( 1+MXLLDA ) = C
         A( 2+MXLLDA ) = C
         A( 3+MXLLDA ) = C
         A( 4+MXLLDA ) = C
*
         A( 1+2*MXLLDA ) = K
         A( 2+2*MXLLDA ) = 2.0D0
         A( 3+2*MXLLDA ) = 5.0D0
         A( 4+2*MXLLDA ) = 6.0D0
*
* Локальная часть матрицы A для процесса (1, 0)
*
      ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
         A( 1 ) = C
         A( 2 ) = C
         A( 3 ) = K
*
         A( 1+MXLLDA ) = C
         A( 2+MXLLDA ) = C
         A( 3+MXLLDA ) = 2.0D0
*
         A( 1+2*MXLLDA ) = C
         A( 2+2*MXLLDA ) = C
         A( 3+2*MXLLDA ) = 5.0D0
*
         A( 1+3*MXLLDA ) = C
```

```
         A( 2+3*MXLLDA ) = C
         A( 3+3*MXLLDA ) = 6.0D0
*
* Локальная часть матрицы A для процесса (1, 1)
*
      ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.1 ) THEN
         A( 1 ) = K
         A( 2 ) = C
         A( 3 ) = 3.0D0
*
         A( 1+MXLLDA ) = C
         A( 2+MXLLDA ) = K
         A( 3+MXLLDA ) = 4.0D0
*
         A( 1+2*MXLLDA ) = 3.0D0
         A( 2+2*MXLLDA ) = 4.0D0
         A( 3+2*MXLLDA ) = 7.0D0
*
      END IF
*
      RETURN
      END
```

Результаты:

```
 Значение  INFO  =   0

 Собственные значения:

  W(1)= -6.D+00
  W(2)=  1.D+00
  W(3)=  1.D+00
  W(4)=  1.D+00
  W(5)=  1.D+00
  W(6)=  1.D+00
  W(7)= 14.D+00

 Собственные векторы:
```

```
Z(1,1)=  -0.8451542547285l6519D-01
Z(2,1)=  -0.16903085094570302 6D+00
Z(3,1)=  -0.25354627641855498 3D+00
Z(4,1)=  -0.33806170189140660 7D+00
Z(5,1)=  -0.42257712736425817 6D+00
Z(6,1)=  -0.50709255283710985 6D+00
Z(7,1)=   0.59160797830996159 1D+00

Z(1,2)=  -0.25522095393728194 7D+00
Z(2,2)=  -0.60960395412412016 2D+00
Z(3,2)=   0.74152708811449619 1D+00
Z(4,2)=   0.80410405466749823 6D-02
Z(5,2)=  -0.10815552822076385 2D+00
Z(6,2)=  -0.40256487206808189 1D-01
Z(7,2)=   0.11102230246251565 4D-15

Z(1,3)=   0.95628463489967419 1D+00
Z(2,3)=  -0.10900233409502861 6D+00
Z(3,3)=   0.21816150602408379 6D+00
Z(4,3)=  -0.54499670216308394 9D-01
Z(5,3)=  -0.11257705065352174 9D+00
Z(6,3)=  -0.10198009177483771 7D+00
Z(7,3)=   0.27755575615628913 5D-16

Z(1,4)=   0.00000000000000000 0D+00
Z(2,4)=  -0.98607566185050007 2D-02
Z(3,4)=  -0.72293482329949268 5D-01
Z(4,4)=   0.89866623399249512 3D+00
Z(5,4)=  -0.23910923933196925 9D+00
Z(6,4)=  -0.36041946318054607 4D+00
Z(7,4)=   0.00000000000000000 0D+00

Z(1,5)=   0.00000000000000000 0D+00
Z(2,5)=  -0.72997777574084488 1D-02
Z(3,5)=  -0.82356449256609579 0D-01
Z(4,5)=   0.49340131022231725 8D-01
Z(5,5)=  -0.75935481820454697 7D+00
Z(6,5)=   0.64351374503640890 2D+00
Z(7,5)=   0.00000000000000000 0D+00
```

```
Z(1,6)=   0.969172365326708696D-01
Z(2,6)=  -0.756567872068086622D+00
Z(3,6)=  -0.540030650666323941D+00
Z(4,6)=   0.105382651046173337D+00
Z(5,6)=   0.259057033086912525D+00
Z(6,6)=   0.219915781663869714D+00
Z(7,6)=   0.222044604925031308D-15

Z(1,7)=  -0.620173672946042337D-01
Z(2,7)=  -0.124034734589208551D+00
Z(3,7)=  -0.186052101883812715D+00
Z(4,7)=  -0.248069469178416879D+00
Z(5,7)=  -0.310086836473021099D+00
Z(6,7)=  -0.372104203767625319D+00
Z(7,7)=  -0.806225774829854913D+00
```

Приложение 2

Головная программа для вызова программы из Приложения 1 с использованием подпрограммы PDELSET для распределения матрицы по решетке процессов.

```
      PROGRAM TDSYEV21
*
*     Тест для программы PDSYEV2
*
      include 'mpif.h'
* Именованные константы - параметры задачи
      INTEGER  LWORK, MAXN, LDA, MAXPROCS, NOUT
      DOUBLE PRECISION   ZERO, MONE
      PARAMETER                 ( LDA = 100, LWORK = 70, MAXN = 100,
*      PARAMETER                 ( LDA = 100, LWORK = -1, MAXN = 100,
     $                                    MAXPROCS = 512, NOUT =6, ZERO = 0.0D
     $                                    MONE = -1.0D+0 )
*
      CHARACTER          UPLO
      PARAMETER          ( UPLO = 'L' )
*
* Локальные переменные
      INTEGER            CTXT, I, IAM, INFO, MYCOL, MYROW, N, NB,
     $                          NPCOL, NPROCS, NPROW, IA, JA, IZ, JZ
*
* Локальные массивы
      INTEGER                     DESCA( 9 ), DESCZ( 9 )
      DOUBLE PRECISION   A( LDA, LDA ), W( MAXN ),
     $                   WORK( LWORK ), Z( LDA, LDA ), WORK1( 7 )
*    $                   WORK( 70 ), Z( LDA, LDA ), WORK1( 7 )
*
* Внешние подпрограммы
      EXTERNAL           BLACS_EXIT, BLACS_GET, BLACS_GRIDEXIT,
     $                   BLACS_GRIDINFO, BLACS_GRIDINIT, BLACS_PINFO,
     $                   BLACS_SETUP, DESCINIT, PDMATINIT, PDLAPRNT,
     $                   PDSYEV2
*
* Выполняемые операторы
```

```
*
* Постановка задачи
*
      N = 7
      NB = 2
      NPROW = 2
      NPCOL = 2
      IA = 1
      JA = 1
      IZ = 1
      JZ = 1
*
* Инициализация пакета BLACS
*
      CALL BLACS_PINFO( IAM, NPROCS )
      IF( ( NPROCS.LT.1 ) ) THEN
         CALL BLACS_SETUP( IAM, NPROW*NPCOL )
      END IF
*
* Инициализация контекста BLACS'а и решетки процессов
*
      CALL BLACS_GET( -1, 0, CTXT )
      CALL BLACS_GRIDINIT( CTXT, 'R', NPROW, NPCOL )
      CALL BLACS_GRIDINFO( CTXT, NPROW, NPCOL, MYROW, MYCOL )
*
* Если процесс не является частью данного контекста,
* переход на конец программы
*
      IF( MYROW.EQ.-1 ) GO TO 20
*
* Инициализация дескрипторов для матриц A и Z
*
      CALL DESCINIT( DESCA, N, N, NB, NB, 0, 0, CTXT, LDA, INFO )
      CALL DESCINIT( DESCZ, N, N, NB, NB, 0, 0, CTXT, LDA, INFO )
*
* Построение матрицы A
*
      CALL PDMATINIT( N, A, IA, JA, DESCA, INFO )
*
```

```
* Печать входных данных
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         WRITE( NOUT, FMT = 92 )
         WRITE( NOUT, FMT = 98 )N, N, NB, NB
         WRITE( NOUT, FMT = 97 )NPROW*NPCOL, NPROW, NPCOL
         WRITE( NOUT, FMT = * ) ' Исходные данные '
         WRITE( NOUT, FMT = 88 ) UPLO
         WRITE( NOUT, FMT = * ) '  DESCA'
         WRITE( NOUT, FMT = 85 )DESCA
         WRITE( NOUT, FMT = * ) '  DESCZ'
         WRITE( NOUT, FMT = 85 )DESCZ
         WRITE( NOUT, FMT = * ) ' Матрица A'
      END IF
   85 FORMAT( / 9I5 /)
*  84 FORMAT( / ' WORK(1) = ', D16.5 /)
   88 FORMAT( / ' UPLO = ', A /)

      CALL PDLAPRNT( N, N, A, IA, JA, DESCA, 0, 0, 'A', NOUT, WORK1 )
*
* Вычисление собственных значений и собственных векторов
*
      CALL PDSYEV2( UPLO, N, A, IA, JA, DESCA, W, Z, IZ, JZ,
     $                            DESCZ, WORK, LWORK, INFO )
*
* Печать результатов
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         WRITE( NOUT, FMT = * ) ' Результаты '
         WRITE( NOUT, FMT = 96 )INFO
*         WRITE( NOUT, FMT = 84 )WORK(1)
         WRITE( NOUT, FMT = * ) ' собственные числа'
         DO 10 I = 1, N
            PRINT *, ' W(', I, ')=', W( I ), ';'
 10    CONTINUE
         WRITE( NOUT, FMT = * ) ' собственные векторы'
      END IF
*
* Печать собственных векторов
```

```
*
      CALL PDLAPRNT( N, N, Z, IZ, JZ, DESCZ, 0, 0, 'Z', NOUT, WORK1 )
*
      CALL BLACS_GRIDEXIT( CTXT )
   20 CONTINUE

      CALL BLACS_EXIT( 0 )

   99 FORMAT( 'W=diag([', 4D16.12, //']);' )

   92 FORMAT( / 'Тест к подпрограмме PDSYEV2' )
   98 FORMAT( / ' Вычисление всех собственных значений и собственных',
     $  ' векторов матрицы  A,' //
     $       ' где A - симметричная матрица ',
     $      I2, ' на ', I2,', разбитая на блоки', I2, ' на ', I2, /)
   97 FORMAT( / ' Запуск на ', I2, ' процессах,',
     $       ' образующих решетку размером ', I2, ' на ', I2 /)
   96 FORMAT( / ' Значение INFO  = ', I6 /)
      STOP
      END
*
      SUBROUTINE PDMATINIT( N, A, IA, JA, DESCA, INFO )
*
* PDMATINIT генерирует и распределяет матрицу A по решетке процессов
*
      INTEGER                          IA, INFO, JA, N
      INTEGER                          DESCA( * )
      DOUBLE PRECISION   A( * )
      INTEGER                          I, J, MYCOL, MYROW, NPCOL, NPROW
      EXTERNAL                        PDELSET
      INTRINSIC                        DBLE
*
      INFO = 0
*
      IF( IA.NE.1 ) THEN
         INFO = -3
      ELSE IF( JA.NE.1 ) THEN
         INFO = -4
      END IF
```

```
*
      DO 2  J = 1, N
      DO 1  I = 1, N
      CALL PDELSET( A, I, J, DESCA, 0.0D0)
   1  CONTINUE
   2  CONTINUE
      DO 4  J = 1, N
      DO 3  I = 1, N
         IF( I.EQ.J ) THEN
            CALL PDELSET( A, I, J, DESCA, 1.0D0)
         END IF
   3  CONTINUE
   4  CONTINUE
      J = N
      DO 5  I = 1, N
      CALL PDELSET( A, I, J, DESCA, DBLE(I) )
   5  CONTINUE
      I = N
      DO 6  J = 1, N
      CALL PDELSET( A, I, J, DESCA, DBLE(J) )
   6  CONTINUE
*
      RETURN
      END
```

Приложение 3

Головная программа для вызова программы из Приложения 1 с использованием подпрограммы PDLAREAD при чтении матрицы из файла.

Содержимое файла с исходной матрицей приведено следом за головной программой.

```
      PROGRAM TDSYEV25
*
*     Тест к подпрограмме   PDSYEV2
*
      include 'mpif.h'
* Именованные константы - параметры задачи ..
      INTEGER               LWORK, NOUT, MEMSIZE, NIN, NPR
      PARAMETER             ( MEMSIZE = 20000, LWORK = 70,
     $                                  NOUT =13, NIN = 11, NPR = 6 )
*
      CHARACTER           UPLO
      PARAMETER          ( UPLO = 'L' )
*
* Локальные переменные
      INTEGER            CTXT, I, IAM, INFO, IPA, IPZ, IPW,
     $                          IPWORK, MYCOL, MYROW, NP, NQ,
     $                          N, NB, NPCOL, NPROCS, NPROW, IA, JA, IZ, JZ
*
* Локальные массивы
      INTEGER             DESCA( 9 ), DESCZ( 9 )
      DOUBLE PRECISION    MEM( MEMSIZE ), WORK1(7)
*
* Внешние подпрограммы
      EXTERNAL           BLACS_EXIT, BLACS_GET, BLACS_GRIDEXIT,
     $                   BLACS_GRIDINFO, BLACS_GRIDINIT, BLACS_PINFO,
     $                   BLACS_SETUP, DESCINIT, PDLAPRNT, PDLAREAD,
     $                   PDLAWRITE, NUMROC, PDSYEV2
*
* Выполняемые операторы
*
* Постановка задачи
*
```

```
      N = 7
      NB = 2
      NPROW = 2
      NPCOL = 2
      IA = 1
      JA = 1
      IZ = 1
      JZ = 1
*
* Инициализация пакета BLACS
*
      CALL BLACS_PINFO( IAM, NPROCS )
*
        IF( NPROCS.LT.1 ) THEN
           CALL BLACS_SETUP( IAM, NPROW*NPCOL )
        END IF
*
* Инициализация контекста BLACS'а и решетки процессов
*
      CALL BLACS_GET( -1, 0, CTXT )
      CALL BLACS_GRIDINIT( CTXT, 'R', NPROW, NPCOL )
      CALL BLACS_GRIDINFO( CTXT, NPROW, NPCOL, MYROW, MYCOL )
*
* Если процесс не является частью данного контекста,
* переход на конец программы
*
      IF( MYROW.GE.NPROW .OR. MYCOL.GE.NPCOL ) GO TO 20
*
* Вычисление размеров локальных частей матрицы  A  на всех процессах
*
      NP = NUMROC( N, NB, MYROW, 0, NPROW )
      NQ = NUMROC( N, NB, MYCOL, 0, NPCOL )
*
* Инициализация дескрипторов для матриц A и Z
*
      CALL DESCINIT( DESCA, N, N, NB, NB, 0, 0, CTXT, MAX(1,NP), INFO)
      CALL DESCINIT( DESCZ, N, N, NB, NB, 0, 0, CTXT, MAX(1,NP), INFO)
*
* Распределение памяти  MEM
```

```
*
* Указатель на первый элемент массива собственных значений
      IPW = 1
* Указатель на первый элемент локальной части матрицы  A
      IPA = IPW + N
* Указатель на первый элемент массива собственных векторов
      IPZ = IPA + DESCA(9)*NQ
* Указатель на рабочий массив
      IPWORK = IPZ + DESCZ(9)*NQ
*
* Чтение из файла и распределение матрицы A
*
      CALL PDLAREAD('tdsyev25.dat', MEM(IPA), DESCA, 0, 0, MEM(IPWORK))
*
* Печать входных данных
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         WRITE( NPR, FMT = 92 )
         WRITE( NPR, FMT = 98 )N, N, NB, NB
         WRITE( NPR, FMT = 97 )NPROW*NPCOL, NPROW, NPCOL
         WRITE( NPR, FMT = * ) ' Исходные данные '
         WRITE( NPR, FMT = 88 ) UPLO
         WRITE( NPR, FMT = * )'   IPW,   IPA,   IPZ,   IPWORK'
         WRITE( NPR, FMT = 94 )IPW, IPA, IPZ, IPWORK
         WRITE( NPR, FMT = * ) '  DESCA'
         WRITE( NPR, FMT = 85 )DESCA
         WRITE( NPR, FMT = * ) '  DESCZ'
         WRITE( NPR, FMT = 85 )DESCZ
         WRITE( NPR, FMT = * ) ' Матрица A'
      END IF

  94  FORMAT( / 4I7 /)
  85  FORMAT( / 9I5 /)
*  84 FORMAT( / ' MEM(IPWORK) = ', D16.5 /)
  88  FORMAT( / ' UPLO = ', A /)

      CALL PDLAPRNT( N, N, MEM(IPA), IA, JA, DESCA, 0, 0, 'A',
     $               NPR, WORK1 )
*
```

```
* Вычисление собственных значений и собственных векторов
*
      CALL PDSYEV2( UPLO, N, MEM(IPA), IA, JA, DESCA, MEM(IPW),
     $              MEM(IPZ), IZ, JZ, DESCZ, MEM(IPWORK), LWORK, INFO)
*
* Печать результатов
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         WRITE( NPR, FMT = * ) ' Результаты '
         WRITE( NPR, FMT = 96 )INFO
*         WRITE( NPR, FMT = 84 )MEM(IPWORK)
         WRITE( NPR, FMT = * ) ' собственные числа'
*
         DO 10  I = 1, N
            PRINT *, ' W(', I, ')=', MEM( IPW+I-1 ), ';'
 10    CONTINUE
         WRITE( NPR, FMT = * ) ' собственные векторы'
      END IF
*
* Печать собственных векторов
*
      CALL PDLAPRNT( N, N, MEM(IPZ), IZ, JZ, DESCZ, 0, 0, 'Z', NPR,
     $               WORK1 )
*
* Запись в файл результатов работы подпрограммы PDSYEV2.f
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         OPEN( NOUT, FILE = 'tdsyev25.res', STATUS = 'UNKNOWN' )
         WRITE( NOUT, FMT = * ) ' собственные числа'
         DO 11  I = 1, N
            WRITE( NOUT, FMT = * ) MEM( IPW+I-1 )
 11    CONTINUE
         WRITE( NOUT, FMT = * ) ' собственные векторы'
      END IF
*
      CALL PDLAWRITE( 'tdsyev25.res', N, N, MEM(IPZ), 1, 1, DESCZ,
     $                0, 0, MEM(IPWORK) )
*
      CALL BLACS_GRIDEXIT( CTXT )
```

```
  20  CONTINUE

      CALL BLACS_EXIT( 0 )

  99  FORMAT( 'W=diag([', 4D16.12, //']);' )

  92  FORMAT( / 'Тест к подпрограмме PDSYEV2' )
  98  FORMAT( / ' Вычисление всех собственных значений и собственных',
     $  ' векторов матрицы  A,' //
     $       ' где A - симметричная матрица ',
     $       I2, ' на ', I2,', разбитая на блоки', I2, ' на ', I2, /)
  97  FORMAT( / ' Запуск на ', I2, ' процессах,',
     $        ' образующих решетку размером ', I2, ' на ', I2 /)
  96  FORMAT( / ' Значение INFO  = ', I6 /)
      STOP
      END
```

Файл с исходной матрицей:

```
       7    7
    0.10000D+01
    0.00000D+00
    0.00000D+00
    0.00000D+00
    0.00000D+00
    0.00000D+00
    0.10000D+01
    0.00000D+00
    0.10000D+01
    0.00000D+00
    0.00000D+00
    0.00000D+00
    0.00000D+00
    0.20000D+01
    0.00000D+00
    0.00000D+00
    0.10000D+01
    0.00000D+00
    0.00000D+00
```

```
   0.00000D+00
   0.30000D+01
   0.00000D+00
   0.00000D+00
   0.00000D+00
   0.10000D+01
   0.00000D+00
   0.00000D+00
   0.40000D+01
   0.00000D+00
   0.00000D+00
   0.00000D+00
   0.00000D+00
   0.10000D+01
   0.00000D+00
   0.50000D+01
   0.00000D+00
   0.00000D+00
   0.00000D+00
   0.00000D+00
   0.00000D+00
   0.10000D+01
   0.60000D+01
   0.10000D+01
   0.20000D+01
   0.30000D+01
   0.40000D+01
   0.50000D+01
   0.60000D+01
   0.70000D+01
```

Ниже приводится содержимое файла с результатами работы подпрограммы PDSYEV2.

```
  собственные значения

  -6.00000000000000
   0.999999999999999
```

```
 1.00000000000000
 1.00000000000000
 1.00000000000000
 1.00000000000000
14.0000000000000

собственные векторы

         7       7

   -0.845154254728516519D-01
   -0.169030850945703026D+00
   -0.253546276418554983D+00
   -0.338061701891406607D+00
   -0.422577127364258232D+00
   -0.507092552837109856D+00
    0.591607978309961480D+00
   -0.111381446794637101D+00
   -0.535086269928834346D+00
    0.819957700161629388D+00
   -0.330304485661413322D-01
   -0.155013470859046570D+00
   -0.618549942121308996D-01
    0.111022302462515654D-15
   -0.984609388674179908D+00
    0.126010706987393587D-01
   -0.987146606522586040D-01
    0.607396657301895460D-01
    0.921782793543347578D-01
    0.919501949235075333D-01
   -0.693889390390722838D-17
    0.000000000000000000D+00
   -0.179127814584487122D-01
    0.896727512209512106D-01
   -0.302109418637490967D+00
    0.802546495342590593D+00
   -0.506247915484824174D+00
    0.000000000000000000D+00
    0.000000000000000000D+00
```

```
-0.71056920164752311 6D-02
 0.66513655304053075 8D-02
-0.84107673387491599 4D+00
 0.23322685895507056 6D-01
 0.54032513224397715 8D+00
 0.00000000000000000 0D+00
-0.84554799128018831 6D-01
 0.81804431065574745 3D+00
 0.45928013656363198 0D+00
-0.14392822362257448 6D+00
-0.21937157764436071 6D+00
-0.20946724152704496 2D+00
-0.22204460492503130 8D-15
-0.62017367294604212 9D-01
-0.12403473458920855 1D+00
-0.18605210188381268 7D+00
-0.24806946917841687 9D+00
-0.31008683647302104 4D+00
-0.37210420376762531 9D+00
-0.80622577482985491 3D+00
```

1. *Антонов А.С.* Параллельное программирование с использованием технологии MPI: Учебное пособие. М.: Изд-во МГУ, 2004. 71 с.
2. *Антонов А.С.* MPI: The Message-Passing Interface. http://parallel.ru/tech/tech_dev/mpi.html
3. *Библиотека* численного анализа НИВЦ МГУ, http://num_anal.srcc.msu.ru/
4. *Воеводин В.В.* Математические модели и методы в параллельных процессах. М.: Наука, 1986. 296 с.
5. *Воеводин В.В.* Математические основы параллельных вычислений. М.: Изд-во МГУ, 1991. 345 с.
6. *Воеводин В.В.* Параллельные структуры алгоритмов и программ. М.: ОВМ АН СССР, 1987. 148 с.
7. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. С.-П.: БХВ-Петербург, 2002. 608 с.
8. *Воеводин В.В., Кузнецов Ю.А.* Матрицы и вычисления. М.: Наука, 1984. 320с.
9. *Воеводин В.В.* Вычислительные основы линейной алгебры. М.: Наука, 1977. 304с.
10. *Дацюк В.Н., Букатов А.А., Жегуло А.И.* Введение в организацию и методы программирования многопроцессорных вычислительных систем (методическое пособие, часть I). Ростов-на-Дону: Изд-во РГУ, 2000. 36 с.
11. *Дацюк В.Н., Букатов А.А., Жегуло А.И.* Среда параллельного программирования MPI (методическое пособие, часть II). Ростов-на-Дону: Изд-во РГУ, 2000. 65 с.
12. *Немнюгин С.А., Стесик О.Л.* Параллельное программирование для многопроцессорных вычислительных систем. С.-П.:: БХВ-Петербург, 2002. 400 с.
13. *Корнеев В.Д.* Параллельное программирование в MPI. Новосибирск: Изд-во СО РАН, 2000. 213 с.
14. *E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mckenney, S. Ostrouchov, and D. Sorensen.* LAPACK Users' Guide. Philadelphia: SIAM, 1995.
15. *L.S. Blackford, J. Choi, A. Cleary, J. Demmej, I. Dhillon, J.J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D.W. Walker, and R.C. Whaley.* ScaLAPACK: A portable linear algebra library for distributed memory computers — design issues and performance // Proceedings of Supercomputing'96, Sponsored by ACM SIGARCH and IEEE Computer Society, 1996. (ACM Order Number:

415962, IEEE Computer Society Press Order Number: RS00126.
http://www.supercomp.org/sc96/proceedings/).
16. http://www.netlib.org/scalapack/index.html/.
17. *J. Choi, J. Dongarra, and D. Walker.* PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subroutines // Practice and Experience. 1996. **8**. 517–535.
18. http://www.netlib.org/scalapack/html/pblas_qref.html/.
19. *J. Dongarra, R. van de Geun, and R.C. Whaley.* Two-dimensional basic linear algebra communication subprograms // Environments and Tools for Parallel Scientific Computing, Advances in Parallel Computing / J. Dongarra and B. Tourancheau, Eds. Vol. 6, Elsevier Science Publishers B.V. 1993. pp. 31–40.
20. http://www.netlib.org/blacs/
21. *G. Golub and C.F. Van Loan.* Matrix Computations. Baltimore: Johns Hopkins University Press, 1996.
22. *C.L. Lawson, R.J. Hanson, D. Kincaid, and F.T. Krogh.* Basic linear algebra subprograms for Fortran usage // ACM Trans. Math. Soft. 1979. **5**. 308–323.
23. http://www.netlib.org/blas/
24. *W. Lichtenstein and S.L. Johnsson.* Block-cyclic dense linear algebra // SIAM J. Sci. Stat. Comput. 1993. **14**. 1259–1288.
25. *L. Prylli and B. Tourancheau.* Efficient block cyclic data redistribution // Lecture Notes in Computer Science. Vol. 1. Heidelberg: Springer, 1996. 155–165.
26. *Fernando K.V. and Parlett B.N.* Accurate singular values and differential QD algorithms // Numer. Math. 1994. **67**, N 2. 191–230.
27. *Форсайт Дж., Малькольм М., Моулер К.* Машинные методы математических вычислений. М.: Мир, 1980.