Объектная модель JSCALA.

О.И. Самоваров, И.В. Арапов, В.В. Бабкова

Аннотация. В статье рассматривается объектная модель JSCALA, реализующая пакет прикладных программ SCALAPACK в среде ParJava. Эта модель позволяет создавать переносимые, масштабируемые, параллельные программы решения задач линейной алгебры в среде ParJava. На примере модели JSCALA показано, как следует включать в среду ParJava высокоуровневые модели параллельного программирования.

1. Введение.

В последнее время широкое распространение получили параллельные архитектуры с распределенной памятью – кластерные системы. Одной из трудностей, препятствующих эффективному использованию таких систем, является отсутствие языковых средств высокого уровня, поддерживающих разработку эффективных параллельных программ.

Сегодня существует достаточно много различных подходов к созданию параллельных программ, но наиболее популярным, ставшим фактически стандартом является интерфейс MPI (Massage Passing Interface) [1].

Модель данных, поддерживаемая MPI, предоставляет программисту средства, позволяющие организовать межпроцессорные взаимодействия. Обычно это примитивы, реализующие синхронные или асинхронные пересылки данных между процессами виртуальной сети. Пересылки могут быть организованы как между отдельной парой процессов, так и между всеми членами группы процессов, объединенных общим контекстом.

Параллельная программа, использующая MPI, разрабатывается в терминах модели передачи сообщений. Это значит, что программисту необходимо решить ряд задач, связанных с распределением данных и распределением вычислений, необходимо также организовать доступ, к удаленным данным, используя примитивы MPI. Однако существует ряд моделей параллелизма, позволяющих рассматривать параллельную программу в определениях той прикладной области, задачи которой она решает. Среди таких моделей можно выделить DVM [2], DPJ [3] и др. Эти модели позволяют разрабатывать параллельные программы в естественной для прикладной области системе обозначений. Например, для DVM, модели предназначенной для решения сеточных задач линейной алгебры – это распределенные массивы, параллельные циклы, решетки процессоров. Для DPJ,

модели параллельной обработки объектов – это распределенные контейнеры, итераторы и параллельные алгоритмы.

В ИСП РАН разрабатывается интегрированная система ParJava [4], поддерживающая разработку масштабируемых параллельных программ на языке Java [5] с использованием интерфейса MPI. Среда ParJava включает в себя графический редактор, отладчик, профилировщик, визуализатор трасс и другие средства отладки и анализа параллельных Java-программ.

Также реализованы и интегрированы в среду ParJava модели высокого уровня DPJ и Java-DVM. Используя соответствующие библиотеки классов, программист имеет возможность разрабатывать параллельные программы, используя различные модели параллелизма. Дальнейшая разработка и реализация моделей высокого уровня параллелизма является одним из важных направлений развития среды ParJava.

Наиболее широкое применение высокопроизводительные системы и параллельное программирование находят при решении вычислительных задач. Среди существующих средств разработки параллельных программ ориентированных на решение задач линейной алгебры можно выделить пакет прикладных программ SCALAPACK [6] созданный на факультете «Computer Science» Университета штата Теннеси под руководством Жака Донгарра.

В статье рассматривается объектная модель JSCALA, которая представляет собой объектную реализацию пакета программ SCALAPACK и предназначена для решения задач линейной алгебры. При разработке модели JSCALA требовалось сохранить оригинальную архитектуру SCALAPACK, но в то же время использовать объектную концепцию проектирования, заложенную в Java.

В разделе 2 представлен краткий обзор пакета SCALAPACK, рассматриваются его архитектура и функциональности. Раздел 3 посвящен описанию объектной реализации пакета SCALAPACK на языке Java – JSCALA. В заключении (раздел 4) делаются выводы относительно перспектив развития среды ParJava.

2. SCALAPACK – пакет программ для решения задач линейной алгебры.

Аббревиатура SCALAPACK расшифровывается как «Scalable Linear Algebra Package», что можно перевести на русский язык как «пакет масштабируемых программ, предназначенный для решения задач линейной алгебры». Функционально пакет SCALAPACK разделен на несколько компонент:

LAPACK (Linear Algebra PACKage) – набор подпрограмм, предназначенный для решения систем линейных уравнений, уравнений метода наименьших квадратов, задач на собственные значения, задач с сингулярными или плохо обусловленными матрицами. Высокая эффективность этих подпрограмм обусловлена тем, что в них используются алгоритмы, которые базируются на вызовах функций библиотеки BLAS. Кроме того, каждая подпрограмма пакета имеет один или несколько платформо-зависимых параметров, влияющих на эффективность ее выполнения.

Эти параметры определяются в процессе инсталляции пакета и используются во время выполнения программы.

- BLAS (Basic Linear Algebra Subprograms) – содержит базовые функции линейной алгебры, такие как вычисление скалярного произведения, умножение матрицы на вектор, умножение матриц и т.п. Как известно эффективность выполнения операций с матрицами, в особенности умножение матриц, в большой степени зависит от особенностей архитектуры вычислительного комплекса. Функции BLAS настраиваются таким образом, чтобы учитывать эти особенности. BLAS можно рассматривать как уровень, обеспечивающий эффективность и переносимость SCALAPACK-алгоритмов.
- PBLAS (Parallel Basic Linear Algebra Subprograms) – библиотека функций, обеспечивающих параллельное выполнение базовых алгоритмов линейной алгебры.
- BLACS (Basic Linear Algebra Communication Subprograms) – библиотека подпрограмм межпроцессорного взаимодействия, ориентированная на решение задач линейной алгебры. Вычислительная модель этой библиотеки базируется на предположении о том, что части массивов и векторов распределяются на одно- или двумерную решетку процессов. Решетка процессов создается и может модифицироваться использованием функций BLACS. Кроме того, BLACS содержит функции, позволяющие организовать синхронные взаимодействия между процессами.

В алгоритмах пакета SCALAPACK часто требуется организовать широковещательный обмен данными или выполнить редукционную операцию на части процессах решетки. В BLACS решетка процессов может быть разделена на подмножества, присвоением каждому процессу своего контекста.

Основной задачей BLACS можно считать обеспечение переносимости коммуникационного уровня пакета SCALAPACK.

На рис. 1. представлена иерархия программных модулей пакета SCALAPACK. Компоненты (программные модули), расположенные в области «Local», реализуют последовательные алгоритмы и используют только локальные данные. Модули, расположенные в области «Global», реализуют параллельные алгоритмы решения задач линейной алгебры. Для их выполнения данные (матрицы и векторы) должны быть предварительно распределены на процессоры вычислительной сети.

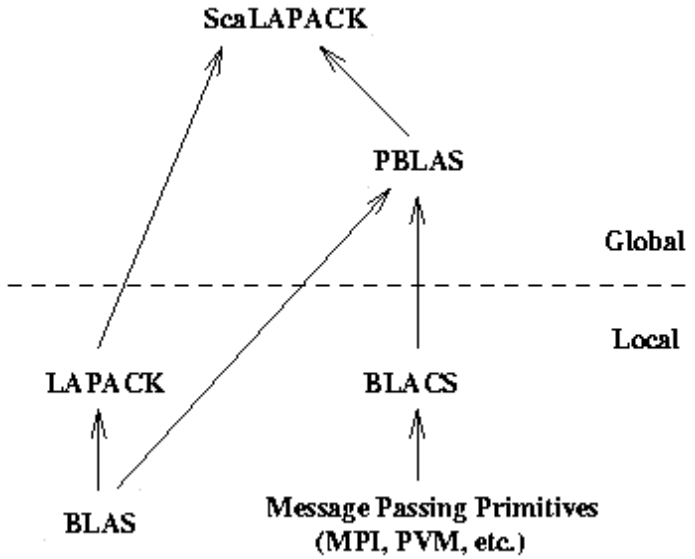

Рис. 1. *Иерархия программных модулей пакета SCALAPACK.*

3. Объектная модель JSCALA.

Существует два способа переноса SCALAPACK на Java. Первый заключается в создании Java-интерфейса для пакета SCALAPACK и реализации этого интерфейса, используя вызовы подпрограмм, входящих в состав SCALAPACK. Второй способ предполагает проектирование новой объектной модели, но в этом случае возникнет необходимость реализации всех подпрограмм SCALAPACK в виде Java-классов и их методов. В данной статье рассматривается второй способ.

Объектная модель JSCALA структурно повторяет SCALAPACK и также разделена на уровни, как показано на рис. 1.

Методы решения задач линейной алгебры собраны в двух классах:

- JSCALADriverRoutines – класс «ведущих» методов, каждый из которых решает законченную задачу, например решение системы линейных уравнений или вычисление собственных значений симметричных матриц. Более подробно методы этого класса описаны в разделе 3.4.1.
- JSCALAComputationalRoutines – класс «вычислительных» методов, предназначенных для решения специальных задач, таких как LU-факторизация, приведение вещественных, симметричных матриц к трехдиагональному виду и т.п. Необходимо отметить, что методы этого

класса используются в тех случаях, когда пользователю для решения его задачи не достаточно стандартных «ведущих» методов, он может сконструировать свой собственный «ведущий» метод, используя «вычислительные» методы. В разделе 3.4.2. содержится подробное описание методов этого класса.

Оба класса наследуются от класса `JSCALA`, содержащего общие поля и методы, такие как:

- jbcomm – объект класса JLAComm, реализует коммуникационный уровень модели и содержит методы межпроцессорного взаимодействия ориентированные на решение задач линейной алгебры.
- jbpla – объект класса JPBaseLA, реализует базовые методы линейной алгебры.

Описанные выше классы являются отображением уровня SCALAPACK в иерархии программных модулей (см. рис. 1).

Коммуникационный уровень модели JSCALA поддерживается классами `JLACommPVM` и `JLACommMPI`. В первом случае в качестве базовой среды межпроцессорного взаимодействия используется пакет PVM (Parallel Virtual Machine) [6], во втором – MPI. Оба класса реализуют интерфейс `JLACommI` описывающий методы коммуникационной подсистемы специализированной на решение задач линейной алгебры. Пользователь имеет возможность, реализовав этот интерфейс самостоятельно, создать собственную версию коммуникационного уровня рассчитанного на иную базовую коммуникационную подсистему. Классы `JLACommPVM` и `JLACommMPI` являются отображением BLACS в иерархии программных модулей SCALAPACK(см. рис. 1). Кроме того, необходимо заметить, что коммуникационный уровень JSCALA привязан к классам ParJava реализующим примитивы MPI и PVM.

Важной частью модели являются классы `JPBaseLALevel1`, `JPBaseLALevel2`, `JPBaseLALevel3`, реализующие базовые методы линейной алгебры. Методы этих классов поддерживают параллельной выполнение алгоритмов и в качестве параметров используют распределенные данные. Подробное описание классов базовой линейной алгебры можно найти в разделе 3.3. Данный уровень объектной модели является отображением BLAS и PBLAS в иерархии программных модулей SCALAPACK (см. рис. 1).

Распределенные данные – матрицы и векторы, объектной модели JSCALA реализуются целым набором классов. Поскольку в Java нет удовлетворительной поддержки многомерных массивов, предлагается использовать специальные классы, которые позволяют создавать двухмерные матрицы различных типов: `IntegerArray`, `FloatArray`, `DoubleArray`. Реализовав интерфейс `ArrayI`, пользователь может создать матрицу собственного типа данных. Для того чтобы определить распределенную матрицу или вектор, пользователь должен создать экземпляр класса `IntegerDArray`, `FloatDArray`, `DoubleDArray`, передав в

качестве параметра ссылку на соответствующий скалярный объект. Класс Descriptor содержит описание способа распределения матрицы, размер распределенной части матрицы, контекст и другую информацию необходимую, для обеспечения доступа к элементам распределенных данных.

На рис. 3. представлена диаграмма классов модели JSCALA.

3.1. Распределенные данные.

Модель JSCALA предполагает, что данные (векторы и матрицы) должны быть распределены на процессоры. Распределенные данные в JSCALA представляются специальными классами `IntegerDArray`, `FloatDArray`, `DoubleDArray`. Используя эти классы, пользователь может создать распределенный массив или вектор нужного типа данных. Эти классы реализуют интерфейс `ArrayI`, используя который пользователь может создать распределенный массив или вектор специального типа.

Общие данные отображаются на локальную память узлов в соответствии с определенным способом распределения. В JSCALA существует три вида распределения данных: блочное, циклическое и блочно циклическое. Блочное распределения данных предполагает, что матрица или вектор делятся пропорционально количеству процессоров, образуя равные блоки которые хранятся в локальной памяти. Циклическое распределение описывает тот случай, когда элементы матрицы распределяются циклически на узлы, образуя смешенные блоки данных каждый из которых хранится на локальном узле. Блочно-циклическое распределение представляет собой совокупность первого и второго способа. В этом случае пользователь может задать размер блоков матрицы, которые будут распределяться циклически на узлы.

3.2. Коммуникационный уровень модели JSCALA.

Коммуникационный уровень модели JSCALA представлен классами `JLAComm`, `JLACommPVM`, `JLACommMPI`, а также интерфейсом `JLACommI`. Этот уровень модели реализует систему межпроцессорных обменов, необходимую для выполнения параллельных алгоритмов линейной алгебры. Кроме того, интерфейсы коммуникационного уровня позволяют использовать эффективные, низкоуровневые средства межпроцессорных обменов и обеспечивают переносимость программ модели JSCALA между широким спектром вычислительных платформ с распределенной памятью.

Система межпроцессорных обменов модели JSCALA основа на следующих ключевых идеях:

Стандартный интерфейс. Одной из основных идей коммуникационного уровня JSCALA является то что, классы и методы, используемые для обеспечения межпроцессорных обменов, могут быть использованы на любой из поддерживаемых платформ.

Интерфейсом между платформами в описываемой модели является `JLACommI`. Текущая версия JSCALA содержат два класса `JLACommPVM` и `JLACommMPI`, реализующих интерфейс `JLACommI`. Эти классы обеспечивают взаимодействие с двумя широко распространенными коммуникационными подсистемами – PVM и MPI.

Решетка процессов. Вычислительная модель JSCALA базируется на предположении о том, что части массивов и векторов распределяются на одно или 2-х мерную решетку процессов. Решетку процессов в данном случае можно представить как абстрактный параллельный компьютер

процессоры которого пронумерованы как $0,1,...,P-1$. Тогда решетка процессоров будет иметь P_τ строк процессоров и P_c столбцов процессоров, где $P_\tau * P_c = P$. Каждый процессор может быть проиндексирован координатами P_τ, P_c, где $0 \le p_\tau < P_\tau$ и $0 \le p_c < P_c$. Пример такого представления процессов приведен на рис. 2, где $P_\tau = 2$ и $P_c = 4$.

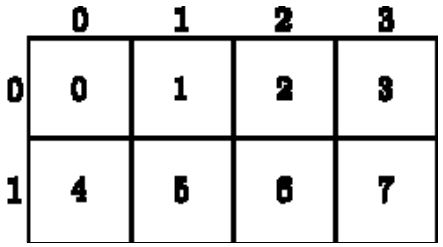

Рис. 2. *Восемь процессов, представленных в виде решетки размерностью* 2×4.

На рис. 2 процессы отображаются на решетку процессоров, используя строковый порядок, иначе говоря, нумерация процессов в решетке производится по строкам слева направо. Существует еще один порядок нумерации процессоров в решетке – нумерация сверху вниз, такой способ отображения называется «по столбцам».

Метод `gridint()` класса `JLAComm` реализует отображение процессов на решетку процессов. По умолчанию используется «строковый» метод отображения. Метод `gridmap` класса `JLAComm` является общим и позволяет пользователю более гибко задавать отображения процессов.

Области операций обменов. В JSCALA поддерживаются так называемые «области» операций обменов. Это значит, что система, использующая линейный массив процессов, имеет область действия операции обмена, равную всем процессам. Однако, если используется двухмерная решетка процессов, существует три области действия операций обмена: `Row` – все процессы в строке; `Column` – все процессы в столбце; `All` – все процессы в решетке.

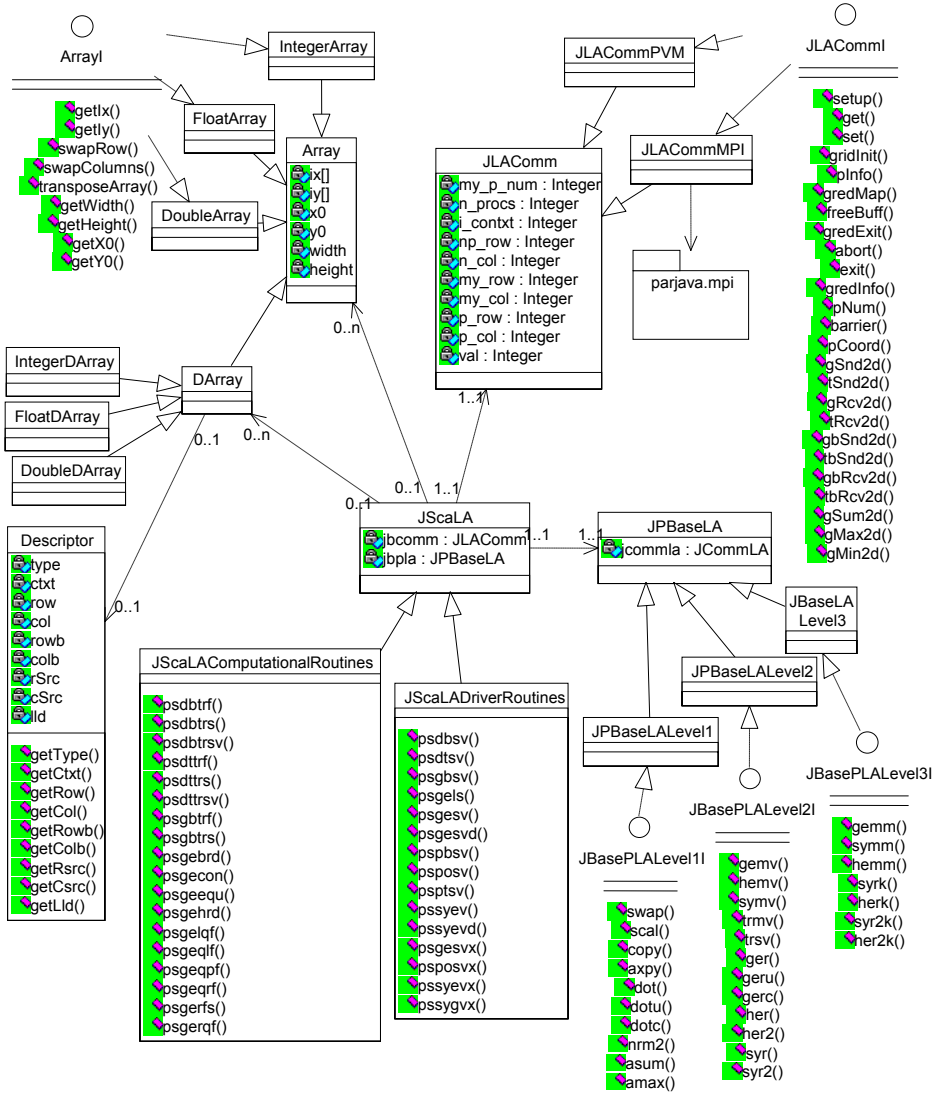

Рис. 3. Диаграмма классов объектной модели JSCALA.

Контекст. В модели JSCALA каждая решетка процессов имеет свой контекст. Также контекст ассоциируется с каждой распределенной матрицей. Использование контекста обеспечивает возможность разделить пространство обмена сообщениями. Это означает, что решетка процессоров обеспечивает безопасные коммуникации даже в то время, когда другая решетка процессоров так же производит операции с

обменом сообщениями. Похожая концепция разделения коммуникационных пространств используется и в MPI. Используемая в JSCALA концепция контекстов позволяет следующее:

- Создавать случайные группы процессов
- Создавать неопределенное число перекрывающихся процессорных решеток
- Разделять решетку процессов так чтобы, обмены производимые между узлами решетки не пересекались.

Класс `JLAComm` модели JSCALA имеет два метода `gridinit()` и `gridmap()`, позволяющие создавать решетки процессоров в собственном контексте. Эти методы возвращают контекст, который представляет собой простое целое число, определяемое внутри класса.

Контекст является расходуемым ресурсом, поэтому он должен быть освобожден, после того как в нем пропала необходимость. Освобождение ресурса осуществляется вызовом метода `gridexit()`.

3.3. Классы базовых методов линейной алгебры.

Модель JSCALA содержит классы, описывающие функции линейной алгебры. Такие как скалярное произведение, умножение матрицы и вектора, умножение матриц. Методы этих классов обеспечивают параллельное выполнения базовых алгоритмов линейной алгебры. Классы этого уровня модели разделены на три типа. Первый обеспечивает операции вида вектор-вектор, второй реализует операции вектор-матрица и третий поддерживает операции матрица-матрица. Ниже перечислены методы классов всех трех типов реализованных в данной версии JSCALA.

3.3.1. Базовые методы первого уровня класса JPBaseLALevel1.

Методы данного класса реализуют базовые параллельные алгоритмы линейной алгебры первого уровня:

- void swap(a,b) – поменять два распределенных вектора,
- double SCAl(a,c) – умножить элементы вещественного, распределенного вектора a на скаляр c,
- void copy(a,b) – скопировать один распределенный вектор в другой,
- void axpy(a,b) – добавить один распределенный вектор к другому,
- double dot(a,b) – внутреннее произведение двух распределенных векторов
- double nrm2(a) – вычислить вторую норму распределенного вектора
- double asum(a) – вернуть сумму абсолютных значений распределенного вектора

- double amax(a) – найти максимальный элемент распределенного вектора a и его индекс

3.3.2. Базовые методы второго уровня класса `JPBaseLALevel2`*.*

Методы данного класса реализуют базовые параллельные алгоритмы линейной алгебры второго уровня:

- double gemv(a,b,A,X) – выполнить операцию:

$$Y = aA*X + bY,$$

где `a` и `b` – скаляры, `X` и `Y` – распределенные векторы, `A` – распределенная матрица.

- double hemv(a,b,A,X) – выполнить операцию:

$$Y = aA*X + bY,$$

где `a` и `b` – скаляры, `X` и `Y` – распределенные векторы, `A` – распределенная Эрмитова матрица.

- double symv(a,b,A,X) – выполнить операцию:

$$Y = aA*X + bY,$$

где `a` и `b` – скаляры, `X` и `Y` – распределенные векторы, `A` – распределенная симметричная матрица.

- double trmv(A) – выполнить операцию:

$$X = A*X$$

где `X` – распределенный вектор, `A` – распределенная верхняя или нижняя треугольная матрица.

- double trsv(A, X) – решить систему уравнений:

$$A*X=b,$$

где `b` и `X` – распределенные векторы, `A` – распределенная верхняя или нижняя треугольная матрица.

- double ger(a, X,Y) – выполнить операцию:

$$A = aX*Y+A,$$

где `a` – скаляр, `X` и `Y` – распределенные векторы, `A` – распределенная матрица.

- double her(a,X) – выполнить Эрмитову операцию первого порядка:

$$A = a*X*conjg(X)+A,$$

где `a` – вещественный скаляр, `X` – распределенный вектор, `A` – распределенная Эрмитова матрица.

- double her2(a,A,X,Y) – выполнить Эрмитову операцию второго порядка:

$$A = a*X*conjg(Y)+conjq(a)*Y*conjg(X)+A$$

где a – вещественный скаляр, X и Y – распределенные векторы, A – распределенная Эрмитова матрица.

- double syr(a,X) – выполнить симметричную операцию первого порядка:

$$A = a*X*X'+A,$$

где a – вещественный скаляр, X – распределенный вектор, A – распределенная симметричная матрица.

- double syr2(a,X,Y) – выполнить симметричную операцию второго порядка:

$$A = a*X*Y' + aY*X'+A,$$

где a – вещественный скаляр, X и Y – распределенные векторы, A – распределенная симметричная матрица.

3.3.3. Базовые методы третьего уровня класса `JPBaseLALevel3`.

Методы данного класса реализуют следующие базовые параллельные алгоритмы линейной алгебры:

- double gemm(a,b,A,B) – выполнить операцию:

$$C=a*op(A)*op(B)+b*C,$$

где a и b – скаляры, A, B, C – распределенные матрицы.

- double symm(a,b,A,B) – выполнить операцию:

$$C=a*A*B+b*C,$$

где a и b скаляры, A – симметричная, распределенная матрица, B, C – распределенные матрицы.

- double hemm(a,b,A,B) – выполнить операцию:

$$C=a*A*B+b*C,$$

где a и b скаляры, A – Эрмитова, распределенная матрица, B, C – распределенные матрицы.

- double syrk(a,b,A) – выполнить симметричную операцию порядка k:

$$C=a*A*A'+b*C,$$

где a и b – скаляры, C – симметричная, распределенная матрица, A - распределенная матрица.

- double herk(a,b,A) – выполнить Эрмитову операцию порядка k:

$$C=a*A*A'+b*C,$$

где a и b скаляры, C – Эрмитова распределенная матрица, A – распределенная матрица.

- double syr2k(a,b,A,B) – выполнить симметричную операцию порядка 2k:

$$C=a*A*B'+aB*A'+b*C,$$

где a и b скаляры, C – симметричная распределенная матрица, A и B – распределенные матрицы.

- double her2k(a,b,A,B) – выполнить Эрмитову операцию порядка 2k:

$$C=a*A*conjg(B')+conjg(a)*B*conjg(A')+b*C,$$

где a и b скаляры, C – Эрмитова распределенная матрица, A и B – распределенные матрицы.

3.4. Классы, реализующие параллельные алгоритмы линейной алгебры.

В модели JSCALA описываются два класса методов реализующих алгоритмы, предназначенные для решения задач линейной алгебры. Первый – `JSCALADriverRoutines`, – содержит методы решения законченных задач, таких как решение системы линейных уравнений, вычисление собственных значений симметричных матриц и др. Этот класс базируется на методы класса `JSCALAComputationalRoutines`. В классе собраны методы, предназначенные для выполнения специальных операций, например LU факторизация, приведение вещественных, симметричных матриц к трехдиагональному виду и пр. Рассмотрим описываемые классы `JSCALADriverRoutines` и `JSCALAComputationalRoutines` более подробно.

3.4.1. Класс «ведущих» методов `JSCALADriverRoutines`.

Здесь и далее методы этого класса мы будем называть «ведущими». Методы класса `JSCALADriverRoutines` решают общие задачи линейной алгебры:

Решение систем линейных уравнений. В классе реализовано два метода решения систем линейных уравнений. Простой метод `psgesv()` – решает систему $AX=B$ факторизацией A и переписыванием B с решением X. И расширенный метод `psgesvx()`, который выполняет некоторые или все из следующих функций: решение $A^T X = B$ или $A^H X = B$ (за исключением случаев, когда A симметричная или Эрмитова матрица); улучшение решения и вычисление передних и задних границ ошибок.

Вычисление собственных значений. Решение задачи нахождения собственных значений сводится к нахождению характеристических чисел λ и соответствующих собственных векторов z != 0, таких что

$$Az = \lambda z, \; A = A^T \qquad \textbf{(2)}$$

где A – вещественная матрица.

Для Эрмитовых матриц мы имеем:

$$Az = \lambda, A = A^T \qquad \textbf{(3)}$$

В обоих случаях λ это вещественные значения. Когда будут найдены все характеристические числа и векторы, можно записать

$$A = Z\Lambda Z^T \qquad \textbf{(4)}$$

где Λ – диагональная матрица элементы, которой характеристические числа, а Z – ортогональная матрица, столбцы которой – характеристические векторы.

В классе `JSCALADriverRoutines` есть метод `pssyev()`, который предназначен для вычисления всех характеристических чисел и векторов семеричной или Эрмитовой матрицы.

Сингулярное разложение. Такое разложение матрицы размерностью m на n можно представить выражением:

$$A = U\Sigma V^T \qquad \textbf{(5)}$$

$$A = U\Sigma V^T \qquad \textbf{(6)}$$

где U и V ортогональны, Σ - это вещественная диагональная матрица размерностью m на n элементы, которой вещественные числа. При этом должно выполняться условие:

$$\sigma_1 \geq \sigma_2 \ldots \sigma_{\min(m,n)} \geq 0 \qquad \textbf{(7)}$$

σ_i это сингулярные значения матрица A, а первая колонка min(m,n) матриц U и V – это левый и правый сингулярные векторы матрицы A. Метод psgesvd() реализует разложение по сингулярным числам общих не симметричных матриц.

Обобщенная симметричная задача собственных значений. Класс содержит метод который обеспечивает вычисление всех собственных значений и характеристических векторов следующих типов задач:

$$Az = \lambda Bz \qquad \textbf{(8)}$$

$$ABz = \lambda z \qquad \textbf{(9)}$$

$$BAz = \lambda z \qquad \textbf{(10)}$$

где A и B это симметричная или Эрмитова матрица и B положительно определена. Для всех задач характеристические значения λ вещественные. Когда матрицы A и B семеричные матрица Z вычисленных собственных значений удовлетворяет условию:

$$Z^T AZ = \Lambda \qquad \textbf{(11)}$$

в случае задач (8) и (10) и условию

$$Z^{-1} AZ^{-T} = I \qquad \textbf{(12)}$$

в случае задачи (9), где Λ это диагональная матрица собственных значений на диагонали. Z также удовлетворяет условию:

$$Z^T BZ = I \qquad \textbf{(13)}$$

в случае задач (8) и (10) и условию

$$Z^T B^{-1} Z = I \qquad \textbf{(14)}$$

в случае задачи (9). Когда A и B Эрмитовы, матрица вычисленных собственных значений Z удовлетворяет условиям:

$$Z^H AZ = \Lambda \qquad \textbf{(15)}$$

в случае задач (8) и (10) и условию

$$Z^{-1} AZ^{-H} = I \qquad \textbf{(16)}$$

в случае задачи (9), где Λ это диагональная матрица собственных значений на диагонали. Z также удовлетворяет условию:

$$Z^H BZ = I \qquad \textbf{(17)}$$

в случае задач (8) и (10) и условию

$$Z^H B^{-1} Z = I \qquad \textbf{(18)}$$

в случае задачи (9).

3.4.2. Класс «вычислительных» методов **JSCALAComputationalRoutines.**

Методы этого класса мы будем называть «вычислительными», поскольку здесь решаются общие задачи линейной алгебры. Необходимо заметить, что с использованием этих методов реализован класс `JSCALADriverRoutines`.

Линейные уравнения. Систему связанных линейных уравнений можно записать как:

$$\Lambda x = b \qquad \textbf{(19)}$$

где A это матрица коэффициентов, b это правая часть, а X решения. В (19) A определено как квадратная матрица порядка n, однако есть методы, в которых A может быть прямоугольной матрицей.

Ортогональная факторизация и линейные задачи наименьших квадратов. JSCALA предлагает методы факторизации прямоугольной матрицы A общего вида размерностью m на n, а также ортогональных матриц и треугольных матриц.

QR факторизация. Наиболее общий и широко известный способ факторизации – это QR факторизация представленная выражением

$$A = Q\begin{pmatrix} R \\ O \end{pmatrix}, \text{ if } m \geq n,$$

где R это треугольная матрица размерностью n на n и Q это ортогональная матрица размерностью m на m. Метод класса `psgeqrf()` реализует вычисления *QR* факторизации.

LQ факторизация. LQ факторизация определяется выражением:

$$A = (L\ 0)Q = (L\ 0)\left(\frac{Q_1}{Q_2}\right) = LQ_1, \text{ if } m \le n,$$

где L это треугольная матрица размерностью m на m, Q это ортогональная матрица размерностью n на n. Q_1 содержит первые m строк матрицы Q, Q_2 содержит остальные n-m строки. Этот вид факторизации поддерживается методом `psgelqf()`.

Другие факторизации. QL и RQ факторизации можно записать как:

$$A = Q\left(\frac{0}{L}\right), \text{ if } m \ge n,$$

и

$$A = (0\ R)Q, \text{ if } m \le n.$$

Эти типы факторизаций вычисляются методами `psgeqle()` и `psgerqf()`. Эти виды факторизаций не так часто используются как рассмотренные выше QR и LQ но могут оказаться полезными при разработке приложений, например для вычисления QR факторизации.

В текущей версии JSCALA реализована лишь часть методов предлагаемых моделью SCALAPACK. Полная версия этих классов – задача дальнейшего развития системы.

4. Заключение.

В данной статье была рассмотрена объектная модель параллельного программирования JSCALA. Модель JSCALA расширяет среду ParJava - интегрированную среду, предназначенную для разработки высокоэффективных параллельных программ. Приводится описание архитектуры модели, описывается диаграмма классов модели.

Объектные модели параллелизма представляют собой технологию, способную обеспечить все этапы разработки эффективных параллельных программ: проектирование, реализацию и отладку. Объектные модели можно рассматривать как средства высокого уровня, которые способны дать ряд преимуществ при разработке параллельных программ.

Литература:

1. MPI: Message Passing Interface Standard, Message Passing Interface Forum, 2000; http://www.mcs.anl.gov/mpi/index.html
2. В.П. Иванников, С.С. Гайсарян, М.В. Домрачев, О.И. Самоваров. “Объектная модель DVM и ее реализация на языке Java.” Вопросы Кибернетики. Приложения системного программирования. Выпуск 4. 1998.

3. В.П. Иванников, С.С. Гайсарян, М.В. Домрачев, В.Ф. Еч, Н.Н. Шталтовная. “Расширение языка Java средствами разработки параллельных программ с распределением по данным с помощью библиотеки классов DPJ” Вопросы Кибернетики. Приложения системного программирования. Выпуск 4. 1998.
4. А.И. Аветисян, И.В. Арапов, С.С. Гайсарян, В.А. Падарян. “Среда ParJava для разработки SPMD-программ для однородных и неоднородных сетей JavaVM.” Труды Института Системного Программирования Российской Академии Наук. Том 2, 2000.
5. JAVA 2 SDK, Standard Edition http://java.sun.com/products/jdk/1.2/
6. SCALAPACK: SCAlable Linear Algebra PACKage. http://www.netlib.org/SCAlapack/
7. С.С. Гайсарян, М.В. Домрачёв, В.Ф. Еч, О.И.Самоваров, А.И. Аветисян. “Параллельное программирование в среде Java для систем с распределенной памятью. Объектные модели параллельного выполнения.” Труды Института Системного Программирования Российской Академии Наук, Том 1, 1999.
8. Rajkumar Buyya (ed.) "High Performance Cluster Computing": Programming and Applications. Vol. 2. Prentice Hall PTR, New Jersey, 1999.