Методика разработки библиотеки шаблонов BLAS для разреженных матриц

Семенов В.А., Тарлапан О.А.

В статье описывается унифицированная реализация пакета BLAS в виде библиотеки шаблонов на языке Си++. Шаблоны параметризуются типами разреженных матричных и векторных операндов, для которых определяются специальные средства навигации по их портретам. Для большинства форматов методика обеспечивает высокую эффективность кода генерируемых библиотек при существенной унификации и портабельности программ.

1. Введение

Идея формирования базового набора процедур, наиболее часто используемых при реализации методов линейной алгебры (ЛА), впервые была сформулирована еще в 1973 году [7]. Пакет, первоначально предназначенный для работы с векторными данными, получил название BLAS (Basic Linear Algebra Subprograms). В него вошли процедуры, реализующие следующие преобразования:

- скалярное произведение двух векторов,
- преобразование Ахру вида $y \leftarrow \alpha \cdot x + y$,
- копирование элементов одного вектора в другой,
- перестановка местами элементов двух векторов,
- умножение вектора на скаляр,
- векторные нормы (евклидова норма, сумма абсолютных значений элементов и максимальное абсолютное значение),
- вращение Гивенса.

Современные версии пакета значительно расширены. В зависимости от алгоритмической сложности в них выделяют процедуры типа “вектор-вектор”, “матрица-вектор”, “матрица-матрица”, объединенные общими названиями BLAS1, BLAS2 и BLAS3 [1,4].

В настоящее время пакет BLAS считается стандартным базовым средством разработки численного программного обеспечения, доступным как в виде портабельных программ, так и в виде библиотек, ориентированных на конкретные аппаратные платформы.

Как правило, пакеты и библиотеки BLAS предназначены для работы с матричными и векторными данными, представленными в каком-либо фиксированном наборе форматов (обычно одном или двух). Примером

может служить библиотека NIST, обеспечивающая работу сразу с несколькими разреженными матричными форматами [2]. Ограниченность данных программ связана, главным образом, с их слабыми инструментальными возможностями, необходимыми для поддержки новых матричных форматов и обеспечения совместимости между ними. В этих случаях расширение пакетов сводится к трудоемкой повторной реализации процедур BLAS, отличающихся лишь типами векторных и матричных операндов.

Вместе с тем, попытки унифицировать программы по отношению к различным матричным типам могут приводить к потере эффективности, что являлось бы крайне нежелательным для BLAS, обычно используемого в качестве вычислительного ядра сложных прикладных программ.

В настоящей работе предлагается методика унифицированной реализации процедур BLAS для произвольных разреженных матричных форматов с использованием техники шаблонов, предусматриваемой современным стандартом ANSI языка C++.

Рассмотрим применение методики на примере реализации библиотеки шаблонов BLAS2.

2. Состав пакета BLAS2

Пакет BLAS2 включает следующие группы операций, реализующие различные формы матрично-векторного умножения:

$$y \leftarrow \alpha \cdot \mathbf{A} \cdot x + \beta \cdot y,$$

$$y \leftarrow \alpha \cdot \mathbf{A}^{\mathrm{T}} \cdot x + \beta \cdot y,$$

$$x \leftarrow \mathbf{T} \cdot x,$$

$$x \leftarrow \mathbf{T}^{\mathrm{T}} \cdot x,$$

различные формы одноранговой модификации:

$$\mathbf{A} \leftarrow \mathbf{A} + \alpha \cdot x \cdot y^{\mathrm{T}},$$

$$\mathbf{H} \leftarrow \mathbf{H} + \alpha \cdot x \cdot x^{\mathrm{T}},$$

$$\mathbf{H} \leftarrow \mathbf{H} + \alpha \cdot (x \cdot y^{\mathrm{T}} + y \cdot x^{\mathrm{T}}),$$

решение треугольных систем:

$$x \leftarrow \mathbf{T}^{-1} \cdot x,$$

$$x \leftarrow \mathbf{T}^{\mathrm{T}^{-1}} \cdot x.$$

где α, β – скалярные переменные, x, y – вектора, $\mathbf{A}$ – прямоугольная матрица, $\mathbf{T}$ – нижне- или верхнетреугольная матрица, $\mathbf{H}$ – симметричная матрица.

Кроме приведенного стандартного набора процедур, некоторые пакеты [1,3] дополнительно включают операции перестановки и переупорядочивания строк и столбцов матрицы, а также различные

процедуры поиска главного элемента, включая процедуры минимизации заполнения при работе с разреженными матрицами. В ряде случаев набор расширяют частными, но более эффективными вариантами упомянутых общих процедур. Например, наряду со стандартным преобразованием одноранговой модификации реализуют частные преобразования Якоби и Хаусхолдера.

Общий набор процедур обычно распространяют и на случаи вещественных матриц с двойной точностью и комплексных матриц. Таким образом, полная реализация пакета BLAS должна включать несколько алгоритмически эквивалентных версий, различающихся, в конечном счете, типами матричных операндов.

3. Возможные подходы к реализации BLAS

Существует несколько методологически различающихся подходов к реализации пакета BLAS.

Первым является традиционный процедурный способ организации пакета в виде автономных процедур, строго ориентированных на конкретные форматы данных. При тщательном программировании подход обеспечивает максимальную эффективность генерируемого кода, однако является достаточно трудоемким и не удовлетворяет требованиям инструментальности при расширении набора операций, поддержке новых матричных форматов или интеграции с прикладными программами. Другим недостатком подхода являются трудности использования пакета в программах, в которых несущественна или нежелательна конкретизация типа матрицы. Например, методы линейной алгебры вполне могли бы реализовываться в самом общем виде без уточнения конкретных типов матриц при определении для них необходимых операций BLAS.

Данную возможность обеспечивает объектно-ориентированный подход, применение которого в данном случае сводится к организации единой матричной классификации и реализации процедур BLAS в виде полиморфных методов наследуемых матричных классов. Вопросы проектирования матричных классификаций достаточно хорошо проработаны [8,9]. Тем не менее, и здесь можно выделить несколько возможных путей реализации процедур BLAS:

- непосредственно в каждом конкретном матричном классе,
- в матричных суперклассах с использованием виртуального доступа к элементам матриц или операций BLAS более низкого уровня,
- в каждом конкретном матричном классе путем перевызова соответствующего шаблона с указанием конкретных типов.

Первый способ по эффективности и трудоемкости реализации эквивалентен процедурному подходу, но вместе с тем обеспечивает возможность манипулирования с абстрактными матричными типами.

Во втором случае затраты на программирование минимальны, поскольку операции BLAS реализуются только на верхних уровнях матричной иерархии и автоматически наследуются всеми производными классами. Для поддержки нового матричного типа достаточно переопределить лишь базовые операции доступа к элементам. Однако такой способ может приводить к крайне низкой эффективности кода за счет частого (по отношению к числу выполняемых элементарных арифметических операций) использования виртуальных операций доступа к элементам матрицы. Применение операций BLAS более низкого уровня в качестве базовых также снижает возможность оптимизации кода.

Важно заметить, что данный способ не исключает предыдущего, поскольку любая операция матричного суперкласса может быть переопределена в производном классе более эффективным образом с учетом математических свойств и особенностей представления данных. Обеспечение гибкого компромисса между производительностью и затратами на программирование в данном случае может рассматриваться как одно из преимуществ эволюционной разработки матричного обеспечения.

В настоящей работе рассматривается третий альтернативный способ реализации пакета BLAS. Вместо операций доступа к элементам матриц предлагается использовать более экономичные операции итерирования ненулевых элементов в портрете произвольной разреженной матрицы. Методы итерирования могут быть достаточно просто и эффективно реализованы в каждом конкретном классе с учетом особенностей представления данных и использоваться в качестве базовых при реализации процедур BLAS. Чтобы исключить виртуальное использование методов итерирования, пакет организуется в виде библиотеки шаблонов функций, параметризуемых типами скалярных, векторных и матричных операндов. При этом операции BLAS реализуются в каждом конкретном матричном классе через вызов соответствующего шаблона с указанием конкретных типов.

И, наконец, третьим подходом является создание классификации элементарных матриц. Операции с ними эквивалентны некоторым матричным преобразованиям BLAS. Примерами элементарных матриц могут служить матрица Фробениуса, эквивалентная процедуре одноранговой модификации, матрица перестановок, эквивалентная процедуре переупорядочения строк или столбцов матрицы. Подход обеспечивает дружественность интерфейса матричных классов, приближая программирование алгоритмов ЛА к традиционной

математической записи. Недостатком подхода является невозможность представления ряда преобразований BLAS в виде эквивалентных операций с элементарными матрицами. Особенности применения данного подхода подробно рассматриваются в [9].

Остановимся более подробно на реализации предложенной методики.

4. Матричный итератор как средство навигации

Обсудим возможность доступа к элементам матриц с использованием матричного итератора.

В самом деле, большинство алгоритмов ЛА не требуют хаотического доступа к элементам матриц. Наиболее употребимым является последовательное сканирование элементов строки, столбца или диагонали. При этом операция вычисления адреса ненулевого элемента может быть существенно упрощена, если известен адрес соседнего с ним элемента, причем это верно как для плотных, так и для разреженных форматов. В случае разреженных форматов можно надеяться на более ощутимый выигрыш, поскольку большинство из них используют массивы или списки элементов, упорядоченные по индексам.

Использование итераторов позволяет автоматически поддерживать разреженность векторных и матричных операндов и, как следствие, сократить объем вычислений за счет исключения операций с нулевыми элементами. Данный аспект является ключевым и занимает центральное место в технологиях разреженных матриц.

С каждым матричным классом будем связывать дружественный ему класс матричного итератора, для которого определим методы навигации по портрету матрицы. Поскольку итератор имеет непосредственный доступ ко всем членам соответствующего матричного класса, методы навигации могут быть реализованы эффективным образом с учетом особенностей представления конкретных структур данных. При этом может быть выработан унифицированный набор методов навигации, применимый к произвольным матричным объектам.

Матричные итераторы, соответствующие различным матричным типам, могут строиться как независимым образом, так и организовываться в единую иерархию наследуемых классов, параллельную матричной. В последнем случае итераторы могут использоваться при обобщенной реализации операций BLAS в матричных суперклассах.

В качестве примера приведем интерфейс класса итератора для матриц, представленных в формате Кнута:

```
struct KnutElement; //basic struct for non-zero element
class  KnutMatrix;  //Knut matrix format
```

```
class KnutMatrixIterator {
  friend class KnutMatrix;
  KnutMatrix *matr;
  KnutElement *ptr;
public:
  KnutMatrixIterator();
  KnutMatrixIterator(KnutMatrix &matr);
  KnutMatrixIterator(KnutMatrix &matr, Number row, Number col);
  ~KnutMatrixIterator();
//
  void Attach(KnutMatrix &matr);
  void Attach(KnutMatrix &matr, Number row, Number col);
  void Detach();
//
  void SetInRow(Number row, Number col);
  void SetInColumn(Number row, Number col);
//
  inline TValue  Element() const;
  inline TValue& Element();
  inline Number RowIndex () const;
  inline Number ColumnIndex () const;
//
  inline Bool While() const;
  inline Bool While(const Block &bl) const;
  inline Bool While(Number imax, Number jmax) const;
  inline Bool WhileInRow() const;
  inline Bool WhileInRow(Number jmax) const;
  inline Bool WhileInColumn() const;
  inline Bool WhileInColumn(Number imax) const;
//
  inline void NextInRow();
  inline void NextInColumn();
};
```

Закрытые данные класса предназначены для привязки итератора к конкретному матричному объекту и элементу, осуществляемой конструкторами и специальными методами ***Attach*** и ***Detach***. Методы ***SetInRow*** и ***SetInColumn*** устанавливают итератор на конкретный элемент матрицы. В случае отсутствия искомого элемента итератор устанавливается на следующий ненулевой элемент в указанной строке или столбце матрицы. Данное правило распространяется на все методы класса.

Все следующие методы реализованы наиболее эффективным образом в виде inline-функций.

Метод ***Element*** обеспечивает доступ к значению текущего ненулевого элемента матрицы. Методы ***RowIndex*** и ***ColumnIndex*** возвращают значения строчного и столбцового индекса текущего элемента.

Различные модификации метода ***While*** предназначены для проверки нахождения итератора в пределах всей матрицы, строки, столбца или в заданном блочном, строчном или столбцовом сегменте.

Методы итерирования ***NextInRow*** и ***NextInColumn*** переустанавливают указатель на следующий элемент в строке или столбце матрицы.

Идентичный набор методов навигации реализуется в аналогичных классах итераторов, соответствующих другим матричным классам. Сформированный набор методов является функционально полным для непосредственной реализации большинства алгоритмов ЛА. Набор мог бы быть расширен методами навигации в других направлениях, в том числе и в направлениях противоположных упомянутым. В виду редкого использования и возможной низкой эффективности для распространенных матричных форматов они не были включены.

5. Библиотека шаблонов BLAS

Техника использования шаблонов в настоящее время получила широкое распространение при программировании стандартных базовых структур данных, некоторых объектов ЛА и анализа [5,6].

Применение техники шаблонов при реализации пакета BLAS имеет целью исключить виртуальное использование методов доступа к элементам векторных и матричных операндов и тем самым обеспечить максимально возможную эффективность генерируемого кода при полной унификации самих программ по отношению к различным векторным и матричным форматам.

В качестве параметров в шаблоны процедур передаются типы скалярных, векторных и матричных операндов, участвующих в преобразованиях. Для реализации самих процедур BLAS должны быть определены также типы итераторов, необходимые для навигации по разреженным векторным и матричным объектам. Чтобы не перегружать список параметров шаблонов, передачу данных дополнительных типов целесообразно осуществлять неявно с основными типами, с которыми они связаны. В следующем разделе приводится пример реализации шаблона с неявной передачей типов.

Применение шаблона процедуры предполагает определение для передаваемых типов необходимых методов и операций, используемых в процедуре. Поскольку мы не требуем, чтобы классы матричных итераторов организовывались в единую иерархию классов с общим интерфейсом, включающим рассмотренный выше набор методов навигации, ответственность за поддержку необходимых операций целиком возлагается на пользователя.

В качестве примера приведем полный список реализованных шаблонов процедур BLAS2. Для экономии места в некоторых описаниях опущено объявление самого шаблона. Используемые обозначения типов соответствуют семантике передаваемых параметров.

Шаблон процедуры

```
template<TValue, TMatrix,TVector1, TVector2>
void Axpy(TValue alpha, const TMatrix& A, const TVector1& x,
          TValue beta, TVector2& y);
```

определяет стандартную матричную процедуру Axpy. Аналогичное преобразование, примененное к матрице, транспонированной относительно заданной, реализует процедура

```
void AxpyT(TValue alpha, const TMatrix& A,
           const Vector1& x, TValue beta, TVector2& y);
```

Следующие процедуры осуществляют умножение соответствующих нижне- и верхнетреугольных сегментов матрицы на вектор:

```
void UpTriangularMultiply(const TMatrix& T, const TVector1& x,
                          TVector2& y);
void UpTriangularTMultiply(const TMatrix& T, const TVector1&  x,
                           TVector2& y);
void LowTriangularMultiply(const TMatrix& T, const TVector1& x,
                           TVector2& y);
```

Процедуры

```
void RankModify(TValue alpha, TMatrix& A, const TVector1& x,
                const  TVector2& y);
void SymmetricRankModify(TValue alpha, TMatrix& A,
                         const TVector&  x);
```

реализуют простую и симметричную одноранговую модификации произвольной матрицы, а процедура

```
void SymmetricRankModifySymmetricMatrix(TValue alpha,
                        TSymmetricMatrix& A,  const TVector& x);
```

– симметричную одноранговую модификацию симметричной матрицы.

Для решения систем уравнений, заданных нижним или верхним треугольным сегментом матрицы, предназначены процедуры

```
void SolveLowTriangular(const TMatrix& T, TVector1& x,
                        const  TVector2& y);
void SolveUpTriangular(const TMatrix& T, TVector1& x,
                       const  TVector2& y);
```

Заметим, что, несмотря на то, что процедуры фактически оперируют с матрицами специальной структуры, в качестве параметра передается ссылка на матрицу общего вида. Данное обстоятельство связано с тем,

что реальное преобразование применяется не ко всей матрице, а только к заданному сегменту.

6. Программирование процедур BLAS с использованием методики шаблонов

Рассмотрим более подробно реализацию шаблонов BLAS с использованием предложенной методики на примере стандартной матричной процедуры Axpy.

```
// Axpy
// y=alpha*A*x+beta*y
// alpha, beta - scalar
// x           - const vector
// y           - vector variable
// A           - const matrix
template<class TValue, class TVector1,
         class TVector2, class TMatrix>
void Axpy(TValue alpha, const TMatrix& A, const TVector1& x,
          TValue beta, TVector2& y) {
  typedef TVector1::iterator  Iterator1;
  typedef TVector2::iterator  Iterator2;
  typedef TMatrix::dvalue     TDValue;
  typedef TMatrix::iterator   MatrixIterator;

  Iterator1 iter_x(x);
  Iterator2 iter_y(y);
  MatrixIterator iter_matrix(A);
  Dimension dim=A.GetDimension();
  Direction d=A.BestDirection();
  if(d==All || d==Right) {
     // dot implementation for row oriented formats
     y.Scal(beta);
     for(Number i=0; i<dim.n; i++) {
        TDValue sum=0;
        // update Matrix iterator
        for(iter_x.Start(), iter_matrix.SetInRow(i,0);
            iter_x.While() && iter_matrix.While();
            iter_matrix.NextInRow()) {
              Number j=iter_matrix.ColumnIndex();
              // Update X iterator
              for(; iter_x.While(j); iter_x.Next())
                ;
              if(iter_x.While() && iter_x.Index()==j)
                sum+=iter_x.Element()*iter_matrix.Element();
        }
        y[i]+=alpha*sum;
     }
  }
  else {
     // axpy implementation for row oriented formats
     y.Scal(beta);
```

```
      for(iter_x.Start(); iter_x.While(); iter_x.Next()) {
         Number j=iter_x.Index();
         TValue var=iter_x.Element()*alpha;
         for(iter_matrix.SetInColumn(0,j); iter_matrix.While();
             iter_matrix.NextInColumn())
           y[iter_matrix.IndexRow()]+=var*iter_matrix.Element();
      }
   }
}
```

В данной функции используются следующие типы переменных:

- базовые типы представления числа с одинарной и двойной точностью TValue, TDValue,
- векторные типы TVector1, TVector2,
- матричный тип TMatrix,
- типы векторных итераторов Iterator1, Iterator2,
- тип матричного итератора MatrixIterator.

Очевидно, что передача всех типов в качестве параметров шаблона является неудобной и избыточной. Например, тип TDValue является одним из предопределенных типов векторных и матричных классов. Желательно ограничить передаваемый набор и за счет итераторных типов, тем более что они применимы только для навигации по дружественным структурам соответствующих векторных и матричных классов. Этого удается добиться за счет описания вспомогательных типов непосредственно в интерфейсах матричных и векторных классов:

```
class  KnutMatrix : public NonSymmetricMatrix {
...
public:
 // typedefs
   typedef Value value;
   typedef DValue dvalue;
   typedef KnutMatrix matrix;
   typedef KnutMatrixIterator iterator;
...
 //
   inline Direction BestDirection() const;
...
};
```

Подобными описаниями снабжаются и векторные классы. После конструирования итераторов определяется направление итерирования, наиболее предпочтительное для навигации. Необходимо заметить, что для большинства матричных форматов затраты на итерирование элементов в разных направлениях могут существенно различаться и тем самым радикально влиять на общую эффективность процедур BLAS. Например, проход по строке матрицы в разреженном строчном формате

эквивалентен просмотру индексов в плотном массиве, связанном с элементами строки, в то время как проход по столбцу сводится к просмотру всех строчных массивов матрицы и обходится значительно дороже.

Поскольку большинство процедур BLAS допускает разнообразные реализации, связанные со строчной или столбцовой обработкой элементов матриц, в процедуры включаются различные алгоритмические варианты, а их выбор осуществляется автоматически на основе определения наиболее эффективного способа итерирования. Приведенная реализация процедуры Axpy включает два варианта, основанных на векторных операциях скалярного произведения и Axpy. Предпочтительный способ итерирования возвращает метод BestDirection. Мы не приводим описания векторного итератора, поскольку он был достаточно подробно рассмотрен в работе [8]. Там же приводились примеры его практического использования.

Обсудим работу матричного итератора на примере внутреннего цикла в столбцовом варианте Axpy. При вызове метода SetInColumn(0,j) итератор устанавливается на первый ненулевой элемент j-ого столбца и далее сканирует столбец методом NextInColumn. Проверку выхода за пределы матрицы осуществляет метод WhileInColumn. Значение текущего элемента определяется методом Element, а его строчный индекс - RowIndex.

Очевидно, что временные затраты на итерирование будут существенно влиять на эффективность всей процедуры. Поскольку все используемые методы матричного итератора реализованы как inline-подстановки, можно надеяться, что генерируемый код окажется близким к коду эквивалентной программы, работающей с конкретными структурами данных. Это утверждение не теряет силы и в случаях, когда матричные итераторы объединены в единую иерархию классов с виртуальными методами навигации, поскольку при генерации версий шаблона компилятору известны конкретные типы параметров и он может осуществить необходимые inline-подстановки. Единственным методом, который может привести к дополнительным затратам, является метод доступа к элементу вектора Element. Однако в практически содержательных случаях в качестве параметров шаблонов используются плотные векторы, доступ к элементам которых все равно осуществляется через inline-подстановку.

В другом варианте процедуры Axpy используется более сложная схема итерирования, в которой скалярная сумма накапливается путем одновременного просмотра строки разреженной матрицы и одного из разреженных векторов.

Проведенное временное тестирование показывает, что для большинства разреженных матричных форматов процедуры библиотеки шаблонов BLAS по эффективности практически не отличаются от процедур, реализованных путем прямого программирования. Наибольшие потери возникают в случае плотных форматов. Возможным объяснением этого являются ограниченные оптимизационные возможности компиляторов для программных циклов с inline-функциями.

7. Заключение

Таким образом, предложена методика унифицированной реализации пакета BLAS в виде библиотеки шаблонов, параметризуемых типами скалярных, векторных и матричных операндов. Методика обеспечивает высокую эффективность генерируемых процедур при полной их унификации по отношению к различным разреженным векторным и матричным форматам.

Методика предполагает организацию специальных итераторных классов для навигации по портретам разреженных объектов. Итераторы обеспечивают унифицированный и более эффективный доступ к ненулевым элементам по сравнению с обычными методами.

Применение библиотеки шаблонов существенно облегчает разработку новых матричных классов, которая сводится к реализации базовых средств поддержки структур данных и средств навигации по ним. При этом необходимые версии процедур BLAS, ориентированные на новые форматы векторных и матричных операндов и их комбинации, могут быть сгенерированы автоматически по имеющимся шаблонам.

В дальнейшем предполагается обобщить шаблоны BLAS на случай блочных матричных операций, а также включить в библиотеку процедуры следующего уровня BLAS3.

Литература.

1. Dodson D.S., Grimes R.G., Lewis J.G. Sparse extensions to the Fortran Basic Linear Algebra Subprograms, ACM Transactions on Mathematical Software, 1991, 17 , 253-263.
2. Remington K.A., Pozo R. NIST Sparse BLAS User's Guide, National Institute of Standards and Technology, July, 1996.
3. Dongarra J.J., Du Croz J., Hammarling S., Hanson R.J. An Extended Set of FORTRAN Basic Linear Algebra Subprograms, ACM Trans. Math. Soft, 1988, Vol. 14, pp. 1-32.
4. Duff I.S., Marrone M., Radicati G., Vittoli C. A set of Level 3 Basic Linear Algebra Subprograms for sparse matrices, RAL-TR-95049 Computing and

Information Systems Department, Atlas Centre, Rutherford Appleton Lab., Oxon OX11 0QX, September 11, 1995.
5. Stepanov A., Lee M. The standard Template Library, Silicon Graphics Inc., Hewlett-Packard Lab., ANSI X3J16-94-0095/ISO WG21NO482, October 31, 1995.
6. Veldhunizen T. Expression Templates, C++ Report, Vol. 7, No. 5, June 1995, pp. 26-31.
7. Hanson R.J., Krogh F.T., Lawson C.L. A proposal for standard linear algebra subprograms, TM 33-660, Jet Propulsion Lab., Pasadena, Calif., Nov. 1973.
8. Семенов В.А., Тарлапан О.А. Технология реализации разреженных матричных классов // Вопросы кибернетики. Приложения системного программирования. - М.: НСК РАН, 1995, с. 164-188.
9. Семенов В.А., Тарлапан О.А. Объектно-ориентированный подход к программированию прямых методов линейной алгебры. // Вопросы кибернетики. Приложения системного программирования. Выпуск 2.- М.: НСК РАН, 1996, с. 147-171.