УДК 519.6

М.В. БЕЛОУС

РАБОТА С СИСТЕМАМИ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ В ПРОГРАММНОМ КАРКАСЕ NADRA-3D

Рассмотрена организация программного кода конечно-элементного решателя, позволяющая использовать различные алгоритмы решения систем линейных алгебраических уравнений без модификации модулей пост-роения матриц и векторов СЛАУ МКЭ.

Введение. Вычислительные схемы решения систем дифференциальных уравнения методом конечных элементов (МКЭ) сводят исходную задачу к (многократному) построению и численному решению систем линейных алгебраических уравнений (СЛАУ). При этом скорость работы программы решения СЛАУ имеет существенное влияние на скорость работы всего алгоритма. Механизмы ускорения вычислений в решателях СЛАУ включают максимальное использование структуры матрицы (ее разреженности, симметричности), использование параллельных вычислений и реализацию алгоритмов для графических ускорителей. В свою очередь программа построения СЛАУ (программная реализация вычислительной схемы МКЭ) должна учитывать требования к структуре входных данных решателя СЛАУ.

В Институте кибернетики имени В.М. Глушкова НАН Украины создан программный каркас Nadra-3D [1] – набор классов и подпрограмм для построения вычислительных схем МКЭ для задач моделирования пространственных процессов изменения напряженно-деформированного состояния, фильтрации воды, теплопроводности.

В статье рассматривается подход к организации программного кода, примененный в этом программном каркасе и позволяющий использовать различные реализации алгоритмов решения слау без модификации кода модулей построения матриц и векторов СЛАУ МКЭ.

Базовые классы подсистемы работы с СЛАУ. Основой подсистемы работы с СЛАУ в программном каркасе Nadra-3D является следующий набор абстрактных классов:

nSlaeMatrix – базовый класс для описания матриц;

nSlaeVector – базовый класс для описания векторов;

nSLAE – базовый класс для описания совокупности матрицы и вектора как СЛАУ;

nSlaeGenerator – класс, порождающий объекты матриц, векторов и СЛАУ;

nSlaeFactory – класс-фабрика, порождающий объекты-генераторы с интерфейсом nSlaeGenerator;

nSlaeSolver – базовый класс решателя СЛАУ;

nSlaeSolverFactory – класс-фабрика, порождающий решатели СЛАУ.

Схема работы модулей программного каркаса с матрицами и векторами показана на рис. 1 (символом * обозначены указатели на объекты). Как видно из схемы, программные модули имеют доступ только к методам, определенным интерфейсами указанных базовых классов, при этом конкретная схема использования объектами оперативной памяти и детали реализации остаются для них недоступными.

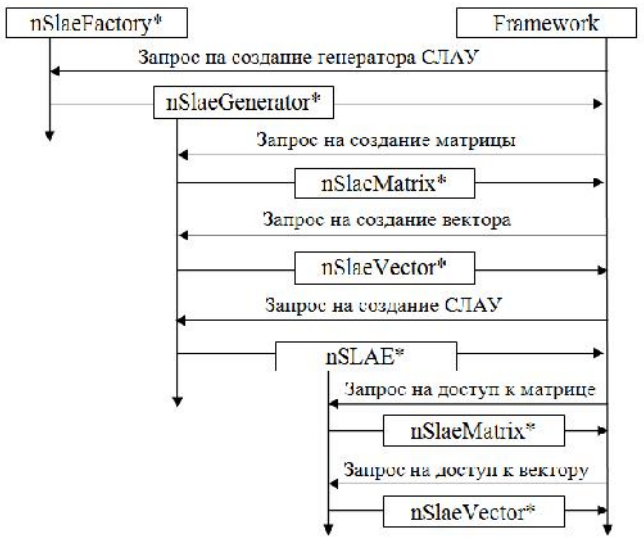
nSlaeFactory*
Framework
Запрос на создание генератора СЛАУ
nSlaeGenerator*
Запрос на создание матрицы
nSlaeMatrix*
Запрос на создание вектора
nSlaeVector*
Запрос на создание СЛАУ
nSLAE*
Запрос на доступ к матрице
nSlaeMatrix*
Запрос на доступ к вектору
nSlaeVector*

РИС. 1. Схема работы программного каркаса с матрицами и векторами

Все операции над матрицами и векторами, а также операции решения СЛАУ выполняются модулями программного каркаса исключительно методами базовых классов.

Загрузив из файла пользовательских настроек код генератора СЛАУ или выбрав код по умолчанию в случае отсутствия настроек система передает его объекту-фабрике nSlaeFactory и получает указатель на конкретный генератор, зарегистрированный за переданным кодом, как указатель на объект класса nSlaeGenerator. Таким образом, пользователю предоставляется возможность выбора и настройки схемы размещения матриц и векторов в оперативной памяти посредством текстового файла (паспорта задачи).

Объект-генератор поддерживает простой программный интерфейс, представленный в листинге 1, и по запросу программных модулей создает объекты, описывающие матрицы, вектора или СЛАУ, и возвращает указатели на них, как указатели на объекты классов nSlaeMatrix, nSlaeVector или nSLAE.

ЛИСТИНГ 1. Интерфейс класса nSlaeGenerator

```
class nSlaeGenerator
{
public:
    nSlaeGenerator(){}
    virtual ~nSlaeGenerator(){}

    virtual nSlaeMatrix* GetMatrix(int dim = 0, int lBhw = 0, int uBhw=0 ) = 0;
    virtual nSlaeVector* GetVector( int dim = 0 ) = 0;
    virtual nSLAE* GetSLAE( int dim = 0, int lBhw = 0, int uBhw = 0 ) = 0;

    virtual void SetParameters( map<string,string> &parameters );
    virtual void GetParameters( map<string,string> &parameters );

    virtual void SetParameters( nParametersSet &parameters ) = 0;
    virtual void GetParameters( nParametersSet &parameters ) = 0;

    int GetSlaeCode(){ return m_slaeCode; }
protected:
    int m_slaeCode;
};
```

На рис. 2 в качестве примера показана схема создания по запросу системы объекта, описывающего матрицу.

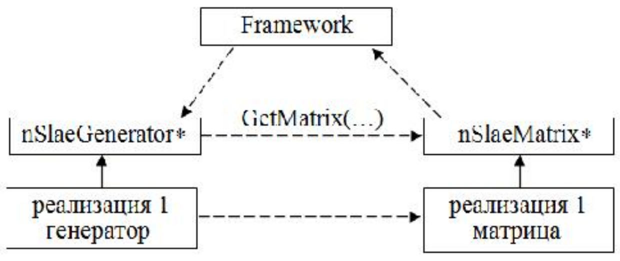

РИС. 2. Схема создания объекта, описывающего матрицу

Добавление в программный каркас новой схемы размещения матриц и векторов в оперативной памяти выполняется путем реализации семейства потомков классов nSlaeMatrix, nSlaeVector, nSLAE, nSlaeGenerator и регистрации этого семейства в объекте-фабрике nSlaeFactory.

Работа с матрицами и векторами. Интерфейс для работы с матрицами и векторами определен в базовых классах nSlaeMatrix и nSlaeVector. Рассмотрим подробнее некоторые основные операции, выполняемые системой над этими объектами.

Операции заполнения элементов матриц и векторов. При обработке сетки конечных элементов на этапах построения СЛАУ, учета краевых условий второго и третьего рода объект вычислительной схемы рассчитывает поправки, которые добавляет к элементам матрицы и вектора. При этом для каждого конечного элемента вычисляется набор таких поправок (элементарная матрица либо элементарный вектор), а их количество определяется количеством узлов конечного элемента и типом решаемой задачи. Для работы с наборами этих поправок в интерфейсах базовых классов определены следующие методы:

```
virtual void nSlaeVector::AllocateValues( int n, int *N, double *F );
virtual void nSlaeVector::AddValues( int n, int *N, double *F );
virtual void nSlaeVector::SubtractValues( int n, int *N, double *F );
virtual void nSlaeMatrix::AllocateFragment(int n, int *N, double **M );
virtual void nSlaeMatrix::AddFragment( int n, int *N, double **M );
virtual void nSlaeMatrix::SubtractFragment(int n, int *N, double **M );
virtual void nSlaeMatrix::AllocateValues(int n,int *rowN,int *colN, double *M);
virtual void nSlaeMatrix::AddValues(int n, int *rowN, int *colN, double *M);
virtual void nSlaeMatrix::SubtractValues(int n,int *rowN,int *colN, double *M);
```

Также определены операции заполнения некоторым значением отдельных строк, столбцов или диагоналей матрицы.

```
virtual void nSlaeMatrix::FillColByValue( int N, double v );
virtual void nSlaeMatrix::FillRowByValue( int N, double v );
virtual void nSlaeMatrix::FillColAndRowByValue( int N, double v );
virtual void nSlaeMatrix::FillDiagonalByValue( int N, double v );
virtual void nSlaeMatrix::FillMatrixByValue( double v );
```

При использовании вышеуказанных операций следует помнить, что, например, операция FillColByValue(…) в симметричных матрицах может, кроме желаемого заполнение столбца, автоматически заполнять соответствующую строку для сохранения симметричности матрицы.

Операции получения и замены строк и столбцов матрицы. Используются при обработке условий первого рода, учета условий сопряжения.

```
virtual bool nSlaeMatrix::GetCol( int N, nSlaeVector *V );
virtual bool nSlaeMatrix::GetRow( int N, nSlaeVector *V );
virtual bool nSlaeMatrix::ReplaceCol( int N, nSlaeVector *V);
virtual bool nSlaeMatrix::ReplaceRow( int N, nSlaeVector *V );
```

Операции вычисления взвешенной суммы строк или столбцов матрицы. Используются, например, при учете условий первого рода.

```
virtual bool nSlaeMatrix::AccumulateColsSum( int n, int *N, double *D,
nSlaeVector *V, bool replace_vector = false );
virtual bool nSlaeMatrix::AccumulateRowsSum( int n, int *N, double *D,
nSlaeVector *V, bool replace_vector = false );
```

Некоторые операции векторной и матрично-векторной алгебры.

```
virtual bool nSlaeVector::AddVector( nSlaeVector *V, double d = 1. );
virtual bool nSlaeVector::SubtractVector(nSlaeVector *V, double d = 1. );
virtual bool nSlaeVector::CloneVector( nSlaeVector *V, double d = 1. );
virtual void nSlaeVector::MultiplyByValue( double d );
virtual bool nSlaeVector::MultiplyByVector( nSlaeVector *V, double &res );
virtual bool nSlaeMatrix::MultiplyByVector(nSlaeVector *V,  nSlaeVector *res );
```

Все перечисленные операции имеют два вида сигнатур. Вышеприведенный вид с использованием интерфейсов nSlaeMatrix, nSlaeVector, а также вид с использованием для передачи и возвращения данных указателей на массив чисел с плавающей точкой. Например:

```
virtual bool nSlaeVector::GetFullContent( int &dim, double *&V );
virtual bool nSlaeVector::SetFullContent( int &dim, double *&V );
virtual bool nSlaeMatrix::GetCol( int N, nSlaeVector *V );
virtual bool nSlaeMatrix::GetCol( int N, int &dim, double *&V );
```

Работа с системами линейных уравнений. В программном каркасе предусмотрен класс nSLAE (листинг 2), описывающий совокупность матрицы и вектора именно как систему линейных алгебраических уравнений.

ЛИСТИНГ 2. Интерфейс класса nSLAE

```
class nSLAE
{
public:
    nSLAE( int dim = 0, int lBhw = 0, int uBhw = 0 );
    virtual ~nSLAE(){}

    virtual nSlaeMatrix *GetMatrix() = 0;
    virtual nSlaeVector *GetVector() = 0;

    int GetSlaeCode(){ return m_slaeCode; }
protected:
    int m_slaeCode;
};
```

Выделение отдельного класса для описания системы уравнений обусловлено тем, что некоторые реализации алгоритмов решения СЛАУ могут использовать схемы размещения матриц и векторов в одном массиве данных. В таких случаях с точки зрения программного каркаса они представляют два различных объекта и работа с ними осуществляется через интерфейсы классов nSlaeMatrix и nSlaeVector, а с точки зрения алгоритма решения – это один объект и работа с ним осуществляется путем предоставления прямого доступа к внутренней структуре данных.

На рис. 3 показаны возможные варианты различной реализации схем использования оперативной памяти, имеющие одинаковый интерфейс nSLAE.

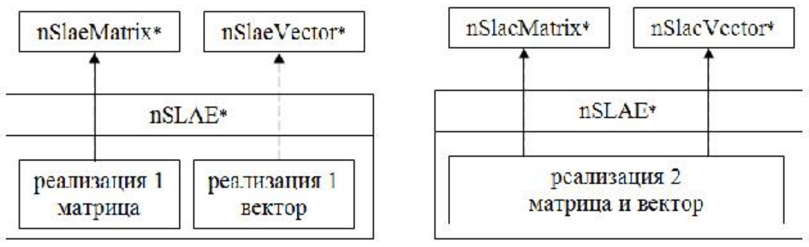

РИС. 3. Варианты реализации классов с интерфейсом nSLAE

Экспорт матриц и векторов в файлы. Сохранение матриц и векторов в файлы осуществляется объектом класса nSlaeSaveLoad. Для каждого типа матрицы или вектора и каждого формата экспорта реализуются специализированные функции сохранения данных с сигнатурами
`bool MatrixSaver(nSlaeMatrix*, FILE*, nMatrixFileFormat)`
и
`bool VectorSaver(nSlaeVector*, FILE*, nVectorFileFormat)`,
которые регистрируются в этом объекте.

При вызове метода
`bool nSlaeSaveLoad::SaveMatrixToFile(nSlaeMatrix*, FILE*, nMatrixFileFormat)` для сохранения матрицы или `bool nSlaeSaveLoad::SaveVectorToFile(nSlaeVector*, FILE*, nVectorFileFormat)` для сохранения вектора, объект nSlaeSaveLoad запрашивает у полученных объектов идентификатор m_slaeCode и ищет в реестре соответствующую функцию сохранения данных. Если она существует, выполняется сохранение данных, в противном случае – в подсистему вывода сообщений передается сообщение об ошибке.

Решение СЛАУ. Базовым объектом для решателей СЛАУ является объект nSlaeSolver, интерфейс которого приведен в листинге 3.

ЛИСТИНГ 3. Интерфейс класса nSlaeSolver

```
class nSlaeSolver
{
public:
    nSlaeSolver();
    virtual ~nSlaeSolver();

    virtual bool Solve( nSlaeMatrix *A, nSlaeVector *B, nSlaeVector *X ) = 0;

    virtual void SetParameters( map<string,string> &parameters );
    virtual void GetParameters( map<string,string> &parameters );

    virtual void SetParameters( nParametersSet &parametersSet ) = 0;
    virtual void GetParameters( nParametersSet &parametersSet ) = 0;

    nSlaeSolverCode GetSolverCode(){ return m_solverCode; }

protected:
    nSlaeSolverCode m_solverCode;
};
```

Каждый наследник этого класса реализует некоторый алгоритм решения СЛАУ и предназначен для работы с матрицами и векторами, хранящимися в оперативной памяти в некотором специализированном формате. То есть объект-решатель СЛАУ может работать не со всеми переданными ему объектами nSlaeMatrix и nSlaeVector, а только с поддерживающими требуемый алгоритмом формат размещения в оперативной памяти. Объект nSlaeSolver определяет свою совместимость с полученными матрицей и вектором сравнением их идентификаторов m_slaeCode и собственного идентификатора m_solverCode.

В случае совместимости данных и алгоритма запускается процедура решения, при этом алгоритму предоставляется прямой доступ к внутренней структуре данных матрицы и вектора (рис. 4), необходимый для эффективной реализации параллельных вычислений.

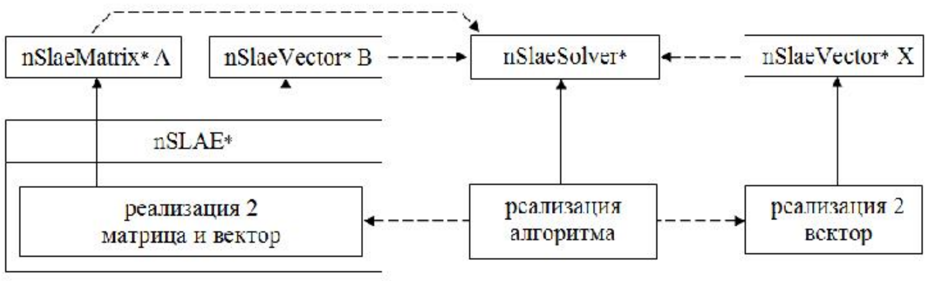

РИС. 4. Схема взаимодействия алгоритма решения и схемы хранения СЛАУ в оперативной памяти

Для одной схемы размещения СЛАУ в оперативной памяти возможна реализация нескольких алгоритмов решения или реализация несколько версий одного алгоритма. Пользователю предоставляется возможность выбора и конфигурации используемого решателя СЛАУ в файле настроек.

Создание системой объекта-решателя, реализующего указанный пользователем алгоритм, выполняется объектом-фабрикой класса nSlaeSolverFactory по запросу с выбранным кодом решателя. Программному каркасу возвращается указатель на созданный решатель как указатель на объект класса nSlaeSolver. На рис. 5 показана схема взаимодействия модулей программного каркаса и подсистемы решения СЛАУ.

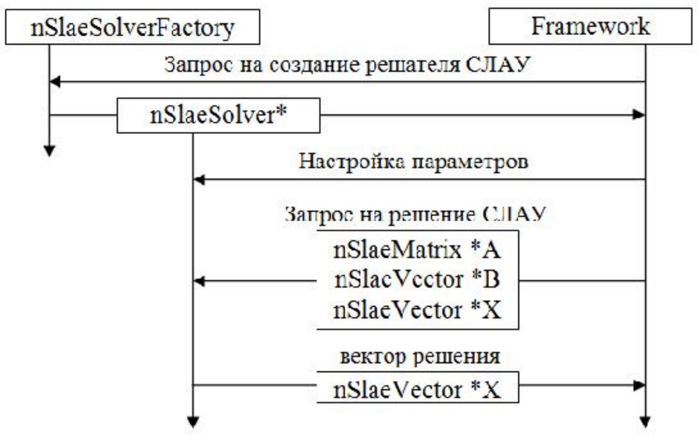

РИС. 5. Схема взаимодействия модулей программного каркаса и подсистемы решения СЛАУ

Добавление в программный каркас нового решателя СЛАУ выполняется путем реализации соответствующего потомка класса nSlaeSolver и регистрации его в объекте класса nSlaeSolverFactory.

С использованием описанного подхода в программный каркас Nadra-3D интегрированы несколько решателей СЛАУ. Далее приведены их краткие характеристики.

Программа решения СЛАУ с симметричной положительно определенной матрицей ленточной структуры. В программный каркас интегрирована созданная в Институте кибернетики имени В.М. Глушкова НАН Украины программа решения СЛАУ с симметричной положительно определенной матрицей ленточной структуры методом LDL^T разложения [2]. Эта программа использует столбцово-слоистую схему размещения СЛАУ в оперативной памяти вычислительного комплекса, причем используемая структура данных является общей для хранения матрицы, вектора правой части и промежуточных результатов работы алгоритма. В структуре хранится нижняя половина симметричной матрицы по столбцам. В процессоре с номером p хранятся столбцы матрицы с номерами $s\cdot(p + j)\ldots s\cdot(p + j + 1)$, где s – размер блока, определенный пользователем, $j = 0\ldots J$, $J = N / (s\cdot P)$, N – размерность матрицы, P – количество процессоров. Работа с этой структурой реализована в наборе классов nSlaeSbrb, nSlaeVectorSbrb, nSlaeVectorSbrbShared, nSlaeMatrixSbrb, nSlaeMatrixSbrbShared, предоставляющих интерфейсы nSlae, nSlaeVector, nSlaeMatrix. Поскольку структура данных является общей для хранения матрицы и вектора правой части, понадобилась реализация двух семейств классов nSlaeSbrb и nSlaeSbrbShared – соответственно для матрицы и вектора, хранящихся в разных массивах данных и в одном массиве данных. Решение СЛАУ выполняется в два этапа – LDL^T факторизация матрицы и нахождение решения полученной треугольной системы. При этом основное время работы алгоритма занимает факторизация матрицы. Если систему необходимо решать многократно и при этом матрица не меняется, то факторизацию необходимо выполнить только один раз. Поскольку после этого этапа на месте исходной матрицы хранится факторизированная матрица, то перед запуском алгоритма необходимо сохранить в кэш столбцы исходной матрицы, которые понадобятся для учета краевых условий первого рода. Алгоритм предназначен для работы с матрицами ленточной структуры, ориентирован на системы с распределенной памятью и имеет довольно сложную внутреннюю структуру хранения СЛАУ. Рекомендуемое соотношение полуширины ленты матрицы, количества процессоров и размера блока выражается формулой [2] $s \cdot P^2 \leq m$, где m – полуширина ленты матрицы.

При использовании алгоритма для систем с матрицами, не удовлетворяющими этому соотношению, время расчетов значительно увеличивается за счет роста количества операций пересылки данных между процессорами и простоя части процессоров в ожидании данных. Кроме того, значительно увеличиваются требования к объему оперативной памяти. В табл. 1 представлены оценки необходимой оперативной памяти при использовании симметричной ленточной схемы хранения матрицы для расчетной сетки тетраэдральных конечных элементов с линейными (в таблице обозначены как сетки класса L) и квадратичными (сетки класса Q) функциями формы при моделировании фильтрации воды.

ТАБЛИЦА 1

Класс сетки	Количество узлов	Максимальная разница номеров узлов	Количество конечных элементов	Оценка объема оперативной памяти
L4	9 282	459	48 120	32 Mb
Q4	9 009	970	5 820	66.7 Mb
L4.5	69 044	1 704	384 720	898 Mb
Q4.5	69 085	4 345	48 120	2.2 Gb
L5.5	416 683	5 898	2 390 880	18.5 Gb
Q5.5	532 413	15 110	384 720	60 Gb
L6	1 026 463	10 228	5 977 200	78.3 Gb
Q6	1 012 121	20 400	736 500	154 Gb

Решатель DSS (Direct Sparse Solver) для разреженных матриц из библиотеки Intel MKL. Предоставляет простой программный интерфейс для нахождения решения СЛАУ [3]. Схема решения, как в вышеописанной программе для СЛАУ с ленточными матрицами, состоит из шагов, типичная последовательность выполнения которых показана на рис. 6.

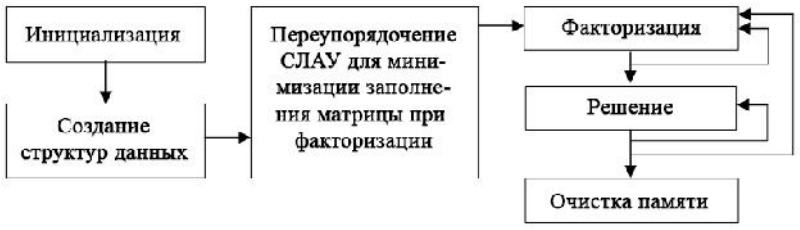
Инициализация
Создание структур данных
Переупорядочение СЛАУ для минимизации заполнения матрицы при факторизации
Факторизация
Решение
Очистка памяти

РИС. 6. Этапы решения СЛАУ

Решатель DSS предполагает использование формата CSR (Compressed Sparse Row) для хранения матрицы СЛАУ в оперативной памяти.

В формате CSR хранятся только ненулевые элементы матрицы, для чего используются три массива данных: values – массив чисел с плавающей точкой для хранения элементов матрицы по строкам; columns – целочисленный массив для хранения номеров столбцов, *i*-й элемент массива содержит номер столбца для *i*-го элемента values; rowIndex – целочисленный массив для хранения индексов начала строк, *j*-й элемент этого массива содержит номер элемента в массиве values, который является первым ненулевым элементом в *j*-й строке матрицы.

Этот формат неудобен на этапе порождения СЛАУ МКЭ, поскольку структура ненулевых элементов матрицы тогда еще неизвестна. Поэтому вместо набора классов для схемы CSR в программном каркасе Nadra-3D реализованы классы nSlaeVectorSparse, nSlaeMatrixSparse, nSlaeSparse для работы с разреженными матрицами в координатном формате, позволяющие выполнять конвертацию внутренних данных в формат CSR перед вызовом решателя DSS.

На разных этапах порождения СЛАУ оптимальными для уменьшения вычислительных затрат являются различные схемы внутреннего хранения элементов матрицы в объектах семейства nSlaeSparse – по строкам или по столбцам, поэтому предусмотрена возможность динамической перестройки внутренней структуры в соответствии с «подсказками», передаваемыми вычислительной схемой. Как и семейство nSlaeSbrb, семейство nSlaeSparse является потомками nSlae.

Решатели СЛАУ в виде отдельных исполняемых файлов. Для работы с решателями СЛАУ, представленными в виде отдельных исполняемых файлов, в программном каркасе Nadra-3D реализован класс nSlaeSolverExternal.

Получив матрицу и вектор СЛАУ как указатели на объекты nSlaeMatrix* A и nSlaeVector *B, этот класс использует методы класса nSlaeSaveLoad для экспорта матрицы и вектора в файлы, после чего генерирует загруженный из файла настроек системный вызов для командной строки с параметрами внешней программы решателя и ждет завершения работы этой программы. После завершения работы внешнего решателя СЛАУ выполняется загрузка найденных решений из файла в вектор nSlaeVector *X с помощью методов класса nSlaeSaveLoad.

Поскольку операции экспорта данных в файлы должны быть реализованы для всех потомков базовых классов матриц и векторов, использовать внешние решатели возможно со всеми поддерживаемыми системой схемами размещения СЛАУ в оперативной памяти. К внешней программе-решателю СЛАУ предъявляется требование уметь импортировать данные из указанного формата.

С использованием интерфейса вызова внешних решателей СЛАУ в каркасе Nadra-3D организована работа с созданными в Институте кибернетики имени В.М. Глушкова НАН Украины программами решения СЛАУ с ленточными [4] и разреженными [5] положительно определенными матрицами на компьютерах гибридной архитектуры (на основе многоядерных CPU и GPU).

При решении трехмерных задач методом конечных элементов целесообразней использовать разреженную схему хранения и алгоритмы решения СЛАУ, ориентированные на ее использование. В табл. 2 приведено сравнение времени работы алгоритмов [2, 5] на вычислительном комплексе Inparcom.

ТАБЛИЦА 2

Параметры СЛАУ dim x bhw	Время работы алгоритма [2]	Время работы алгоритма [5]
69 044 x 1 704	17 сек	4.75 сек.
69 085 x 4 345	1 мин. 39 сек.	7.82 сек.
416 683 x 5 898	19 мин. 43 сек.	1 мин. 29 сек.
532 413 x 15 110		3 мин. 21 сек.
1 026 463 x 10 228		9 мин. 10 сек.
1 012 121 x 20 400		16 мин.

Выводы. Рассмотренный подход к работе с системами линейных алгебраических уравнений обеспечивает гибкую настройку решателя СЛАУ в составе пакета конечно-элементного моделирования в зависимости от имеющихся библиотек и технических характеристик вычислительного комплекса. Использование четко определенных программных интерфейсов для классов, описывающих элементы СЛАУ, позволяет разделить программную реализацию вычислительных схем метода конечных элементов и программную реализацию алгоритмов решения СЛАУ (с учетом различных специализированных схем использования оперативной памяти). Это предоставляет возможность независимого развития указанных подсистем.

М.В. Білоус

РОБОТА З СИСТЕМАМИ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ У ПРОГРАМНОМУ КАРКАСІ NADRA-3D

Розглянута організація програмного коду скінченно-елементного розв’язувача, яка дозволяє використовувати різні алгоритми знаходження розв’язків систем лінійних алгебраїчних рівнянь без модифікації модулів побудови матриць та векторів.

M.V. Bilous

AN OPERATION WITH THE SYSTEMS OF LINEAR EQUATIONS
IN THE SOFTWARE FRAMEWORK NADRA-3D

A technology of integration of different algorithms for solving the systems of linear equations in the finite-element framework is discussed.

1. *Белоус М.В.* Конечно-элементный решатель Надра-3D. Материалы II Международной конференции *Кластерные вычисления*. Львов, 3 – 5 июня 2013. С. 40 – 47.
2. *Попов А.В., Химич А.Н.* Параллельный алгоритм решения системы линейных алгебраических уравнений с ленточной симметричной матрицей. *Компьютерная математика*. 2005. № 2. С. 52 – 59.
3. https://software.intel.com/en-us/node/521696
4. *Баранов А.Ю., Белоус М.В., Сергиенко И.В., Химич А.Н.* Гибридные алгоритмы решения линейных систем для конечно-элементного моделирования процессов фильтрации. *Кибернетика и системный анализ*. 2015. Том 51, № 4. С. 112 – 120.
5. *Хіміч О.М., Сидорук В.А.* Плитковий гібридний алгоритм факторизації розріджених блочно-діагональних матриць з обрамленням. *Компьютерная математика*. 2016. № 1. С. 72 – 79.

Получено 28.04.2017

Об авторе:

Белоус Максим Викторович,
кандидат физико-математических наук
Института кибернетики имени В.М. Глушкова НАН Украины.