Нижегородский государственный университет им. Н.И. Лобачевского
Факультет вычислительной математики и кибернетики

Образовательный комплекс «Параллельные численные методы»

Лабораторная работа
Решение систем линейных алгебраических уравнений с разреженной матрицей прямыми методами средствами библиотеки MKL PARDISO

Сафонова Я.Ю., Кустикова В.Д.

При поддержке компании Intel

Нижний Новгород
2011

Содержание

ВВЕДЕНИЕ **4**

1. МЕТОДИЧЕСКИЕ УКАЗАНИЯ **5**

1.1. Цели и задачи работы 5
1.2. Структура работы 5
1.3. Тестовая инфраструктура 6
1.4. Рекомендации по проведению занятий 6

2. НЕКОТОРЫЕ ВОЗМОЖНОСТИ INTEL® MATH KERNEL LIBRARY PARDISO **7**

2.1. Решение системы линейных алгебраических уравнений с помощью разложения Холецкого 7
2.2. Типы поддерживаемых матриц, CRS формат хранения разреженных матриц 9
2.3. Параметры вызова функции PARDISO 11
2.4. Выделение этапа переупорядочивания матрицы с использованием PARDISO. Пример вызова 18
2.5. Факторизация матрицы и решение системы с факторизованной матрицей с помощью PARDISO. Пример вызова 20
2.6. Дополнительный способ вызова (DSS интерфейс) 20

3. НЕКОТОРЫЕ ВОЗМОЖНОСТИ ПРЯМОГО РЕШАТЕЛЯ СЛАУ В СОСТАВЕ INTEL® MATH KERNEL LIBRARY LAPACK **22**

3.1. Типы поддерживаемых матриц, способ хранения ленточных матриц 22
3.2. Параметры вызова функции dpbtrs 23

4. СТАЦИОНАРНАЯ ЗАДАЧА ВЫЧИСЛЕНИЯ РАСПРЕДЕЛЕНИЯ ТЕМПЕРАТУР В ПЛАСТИНЕ **25**

5. ВЫЧИСЛИТЕЛЬНАЯ СХЕМА **26**

6. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ **28**

6.1. Вспомогательные функции 31
6.2. Последовательная версия №1. Реализация функции решения СЛАУ с использованием последовательного MKL PARDISO 33
6.3. Последовательная версия №2. Реализация функции решения СЛАУ с использованием прямого решателя из MKL LAPACK 37
6.4. Параллельная версия. Реализация функции решения СЛАУ с использованием параллельного MKL PARDISO 39

7. СРАВНЕНИЕ ЭФФЕКТИВНОСТИ РЕШЕНИЯ ЗАДАЧИ ПРИ ИСПОЛЬЗОВАНИИ MKL PARDISO И MKL LAPACK 41

8. АНАЛИЗ МАСШТАБИРУЕМОСТИ ПРИЛОЖЕНИЯ ПРИ ИСПОЛЬЗОВАНИИ MKL PARDISO ДЛЯ РЕШЕНИЯ СЛАУ 45

9. КОНТРОЛЬНЫЕ ВОПРОСЫ 49

10. ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ 49

11. ЛИТЕРАТУРА 50

11.1. Использованные источники информации 50
11.2. Дополнительная литература 50
11.3. Ресурсы сети Интернет 50

Введение

> Библиотеки – гардеробы, из которых умелые люди могут извлекать кое-что для украшения, многое – для любопытства, и еще больше для употребления.
>
> *Дж. Дайер*

Проблема решения системы линейных алгебраических уравнений вида $Ax = b$, где A – матрица размера $n \times m$, b – известный вектор размера n, а x – вектор размера m, является центральной при решении многих задач математической физики [1, 4]. *Решением* системы указанного вида является вектор x, при подстановке которого в систему получается верное тождество. Система вида $Ax = b$ называется *совместной*, если она имеет хотя бы одно решение.

Множество численных методов решения систем алгебраических уравнений (СЛАУ) делятся на прямые (точные) и итерационные методы.

В данной лабораторной работе остановимся на применении некоторых прямых методов решения СЛАУ, т.к. прямые методы позволяют получить точное решение системы за конечное число арифметических действий, а в большинстве реальных приложений требуется найти именно точное решение. Примерами прямых методов являются метод Гаусса, LU-метод, метод Холецкого и другие [1, 2, 3]. Следует отметить, что программная реализация прямых методов приводит к решению с некоторой погрешностью, т.к. все арифметические операции над переменными с плавающей точкой выполняются с округлением. В зависимости от свойств матрицы исходной системы эти погрешности могут достигать значительных величин.

Выбор того или иного прямого метода при решении конкретной задачи во многом определяется «типом» этой задачи [2]. В некоторых случаях необходимо выполнить однократное решение СЛАУ $Ax = b$, в других – требуется найти решение той же системы, но при разных значениях правой части. Одним из определяющих факторов при выборе метода является количество выполняемых операций при решении задачи. Также влияние на выбор оказывает вид матрицы A. Специальные методы решения СЛАУ с ленточными или разреженными матрицами, как правило, имеют меньшую вычислительную сложность по сравнению с методами, которые работают с матрицами общего вида.

На данный момент существует много открытых и коммерческих математических библиотек и пакетов [8], которые, в частности, позволяют решать СЛАУ прямыми методами. В настоящей лабораторной работе будут рассмотрены некоторые возможности библиотеки Intel® Math Kernel Library (MKL) [9] для решения СЛАУ с разреженными и ленточными

матрицами на примере задачи определения распределения температур в прямоугольной пластине.

1. Методические указания

1.1. Цели и задачи работы

Цель данной лабораторной работы – изучение некоторых методов решения СЛАУ с разреженной матрицей на примере стационарной задачи распределения температур в пластине прямоугольной формы с заданным температурным режимом на границах пластины.

Данная цель предполагает решение следующих основных задач:

1. Изучение возможностей пакета Intel® MKL PARDISO.
2. Разработка инфраструктуры для проведения массовых экспериментов.
3. Разработка последовательной и параллельной функций решения СЛАУ с использованием Intel® MKL PARDISO.
4. Разработка функции решения СЛАУ с ленточной матрицей с помощью пакета Intel® MKL LAPACK.
5. Сравнение эффективности решения задачи с использованием последовательных решателей Intel® MKL PARDISO и Intel® MKL LAPACK.
6. Анализ масштабируемости приложения при использовании параллельного решателя Intel® MKL PARDISO, анализ масштабируемости этапов факторизации и переупорядочивания.

1.2. Структура работы

В работе рассматривается стационарная задача распределения температур в прямоугольной пластине с заданным температурным режимом на границах пластины. Распределение температур описывается дифференциальным уравнением в частных производных второго порядка, решение которого сводится к решению системы линейных алгебраических уравнений с блочной пятидиагональной матрицей. В процессе выполнения работы предлагается изучить метод Холецкого и применить его реализацию из библиотеки MKL PARDISO для решения стационарной задачи распределения температур в пластине. Наряду с этим предполагается разработать реализацию с использованием MKL LAPACK. Сравнить эффективность решения задачи с помощью MKL PARDISO и MKL LAPACK. Провести анализ масштабируемости приложения при использовании параллельного решателя MKL PARDISO.

1.3. Тестовая инфраструктура

Вычислительные эксперименты проводились с использованием следующей инфраструктуры (табл. 1).

Таблица 1. Тестовая инфраструктура

Процессор	2 четырехъядерных процессора Intel® Xeon E5520 (2.27 GHz)
Память	16 Gb
Операционная система	Microsoft Windows 7
Среда разработки	Microsoft Visual Studio 2008
Компилятор, профилировщик, отладчик	Intel® Parallel Studio XE 2011
Библиотеки	Intel® Math Kernel Library (в составе Intel® Parallel Studio XE 2011)

1.4. Рекомендации по проведению занятий

Для проведения занятий предлагается следующая последовательность действий:

1. Обсудить понятие разреженной матрицы и способы хранения разреженных матриц. Подробнее рассмотреть CRS формат хранения разреженных матриц.
2. Обсудить понятие ленточной матрицы и способы хранения ленточных матриц.
3. Напомнить вводную информацию описательного характера о методах решения СЛАУ с разреженными и ленточными матрицами.
4. Напомнить общую схему метода Холецкого.
5. Рассмотреть базовые возможности решателей MKL PARDISO и MKL LAPACK.
6. Рассмотреть содержательную постановку стационарной задачи распределения температур в прямоугольной пластине с заданным температурным режимом на границах пластины, а также сведение данной задачи к решению СЛАУ с пятидиагональной матрицей.
7. Разработать тестовую инфраструктуру для сведения стационарной задачи распределения температур к решению СЛАУ (функции формирования матрицы системы, правой части).

8. Разработать последовательную функцию решения СЛАУ с использованием MKL PARDISO.
9. Разработать последовательную функцию решения СЛАУ с использованием прямого решателя из MKL LAPACK.
10. Разработать параллельную функцию решения СЛАУ с использованием MKL PARDISO.
11. Сравнить эффективности решения стационарной задачи распределения температур при использовании последовательных решателей MKL PARDISO и MKL LAPACK.
12. Проанализировать масштабируемость приложения при использовании параллельного решателя MKL PARDISO.

2. Некоторые возможности Intel® Math Kernel Library PARDISO

В данном разделе рассматриваются некоторые базовые возможности решателя MKL PARDISO, а также предлагается описание схемы решения СЛАУ, реализованной в указанном решателе, с помощью разложения Холецкого. Наряду с этим приводятся примеры использования решателя MKL PARDISO.

2.1. Решение системы линейных алгебраических уравнений с помощью разложения Холецкого

Для решения СЛАУ с симметричной положительно определенной[1] матрицей применяют метод Холецкого, который позволяет разложить матрицу A в виде произведения $U^T U$, где U - верхнетреугольная матрица. При этом решение исходной системы можно свести к решению двух систем с помощью введения нового неизвестного вектора y. Для этого подставим разложение матрицы A в исходную систему:

$$Ax = U^T Ux = b.$$

Обозначим через вектор y произведение матрицы U на вектор x. В результате получаем следующие системы линейных алгебраических уравнений:

$$U^T y = b,$$

$$Ux = y.$$

[1] Матрица A положительно определена, если для любого ненулевого вектора X выполняется условие $(AX)X^T > 0$.

Алгоритм нахождения разложения эквивалентен алгоритму Гаусса для симметричных матриц и содержит шаги исключения ненулевых элементов [5]. Когда матрица A является разреженной, исключение может порождать новые ненулевые элементы. В этом случае говорят, что в матрице U возникает $заполнение$. Заполнение зависит от структуры исходной матрицы. В некоторых задачах заполнение является достаточно большим, т.е. верхний треугольник матрицы U становится плотным. Чтобы сохранить разреженность разложения, используют $переупорядочивание$ – умножение исходной матрицы A на матрицу перестановок P, а именно PAP^T. Переупорядочивание изменяет портрет[2] исходной матрицы с помощью симметричных перестановок строк и столбцов, но сохраняет ее знакоопределенность [5], что позволяет применять разложение Холецкого для новой матрицы. Решение системы $Ax = b$ с разреженной симметричной положительно определенной матрицей сводится к решению эквивалентной системы $(PAP^T)Px = Pb$ с точностью до перестановки вектора x. В общем случае задача поиска матрицы перестановок, оптимальной с точки зрения уменьшения заполнения, NP-полна, но существуют различные эвристические алгоритмы, которые позволяют находить оптимальную матрицу перестановок для узкого класса задач.

MKL PARDISO использует разложение Холецкого для решения СЛАУ. Решатель системы $Ax = b$ с положительно определенной матрицей A выполняет следующую последовательность действий:

1. Переупорядочивание матрицы A вида PAP^T.
2. Нахождение разложения U для переупорядоченной матрицы $U^TU = PAP^T$. На данном этапе выделяется символическая и численная фазы. Первая фаза формирует портрет матрицы U и выделяет память, необходимую для ее хранения, вторая – определяет значения ненулевых элементов.
3. Решение треугольных систем. В результате первого и второго шагов требуется решить систему $U^TUPx = Pb$. При введении дополнительного вектора y она распадается на две системы $U^Ty = Pb, UPx = y$. Решение систем соответственно называют *прямой* и *обратной подстановками*.

Отметим, что разработчики PARDISO поддерживают решение СЛАУ и с другими типами матриц, основываясь на их разложении. В частности при решении СЛАУ с симметричной неположительно определенной матрицей схема работы решателя такая же, но разложение переупорядоченной матрицы выполняется в виде U^TDU, где D – блочно-диагональная матрица, U – верхнетреугольная матрица.

[2] Портретом матрицы называется схема расположения ненулевых элементов.

Преимущество использования метода Холецкого при решении СЛАУ состоит в том, что нахождение искомого вектора x выполняется за время $O(n^2)$, где n – порядок системы. Указанная трудоемкость достигается за счет треугольного вида матриц, в то время как другие прямые методы решения СЛАУ, например, метод Гаусса дают кубическую сложность. Метод Холецкого также показывает хорошие временны**е** результаты при решении систем со многими правыми частями или многих систем с матрицами, имеющими одинаковые портреты, т.к. из схемы видно, что в этих случаях разложение матрицы достаточно найти один раз.

Описанная схема решения СЛАУ не обладает большим внутренним параллелизмом в силу определенной зависимости данных в алгоритме построения разложение Холецкого [10]. Поэтому при анализе масштабируемости реализации указанной схемы с использованием параллельного MKL PARDISO нельзя ожидать значительного ускорения.

2.2. Типы поддерживаемых матриц, CRS формат хранения разреженных матриц

MKL PARDISO поддерживает решение СЛАУ с девятью различными типами матриц:

- Матрицы с симметричным портретом и действительными или комплексными элементами.
- Симметричные положительно определенные матрицы с действительными элементами.
- Симметричная матрица с неизвестной знакоопределенностью и действительными элементами.
- Симметричные матрицы с комплексными коэффициентами.
- Несимметричные матрицы с действительными или комплексными элементами.
- Эрмитовы[3] положительно определенные матрицы.
- Эрмитовы матрицы с неизвестной знакоопределенностью.

Для хранения разреженной матрицы в MKL PARDISO применяется широко распространенный формат под названием CSR (Compressed Sparse Rows) или CRS (Compressed Row Storage), который использует три массива. Первый массив хранит значения элементов построчно (строки рассматриваются по порядку сверху вниз), второй – номера столбцов для каждого элемента, а третий – индексы первых элементов каждой строки.

Отметим, что если количество элементов массива **`RowIndex`** равно N + 1, то i-ый элемент массива **`RowIndex`** указывает на начало i-ой строки. При

[3] Эрмитова матрица – квадратная матрица с комплексными элементами, удовлетворяющая соотношению $A^T = \bar{A}$, где $\bar{A}$ - матрица, чьи элементы сопряжены элементам исходной матрицы.

этом элементы строки *i* в массиве **Value** находятся по индексам от **RowIndex[i]** до **RowIndex[i + 1] – 1** включительно. Исходя из этого, обрабатывается случай пустых строк, а также добавляется «лишний» элемент в массив **RowIndex** – устраняется особенность при доступе к элементам последней строки. Данный элемент хранит номер последнего ненулевого элемента матрицы плюс 1, что соответствует количеству ненулевых элементов NZ.

Оценим объем памяти, необходимый для хранения разреженной матрицы в формате CRS. Очевидно, что объем памяти определяется количеством байт, которое требуется для хранения трех указанных массивов: $M = 8\,NZ + 4\,NZ + 4\,(N + 1) = 12\,NZ + 4\,N + 4$.

Ниже приведен пример хранения некоторой матрицы в формате CRS.

A

1				2	
		3	4		
			8		5
	7	1			6

Структура хранения:

1	2	3	4	8	5	7	1	6

Value

0	4	2	3	3	5	1	2	5

Col

0	2	4	4	6	6	9

RowIndex

Рис. 1. Формат хранения CRS

Заметим, что доступ к элементам строки в формате CRS организуется крайне просто.

Операции вставки/удаления элементов приводят к перепаковкам, что является недостатком рассмотренного представления.

В работе будем использовать следующую структуру данных:

```
struct crsMatrix
{
  int N;  // Размер матрицы (N x N)
  int NZ; // Кол-во ненулевых элементов

  // Массив значений (размер NZ)
  double* Value;
  // Массив номеров столбцов (размер NZ)
  int* Col;
  // Массив индексов строк (размер N + 1)
  int* RowIndex;
};
```

2.3. Параметры вызова функции PARDISO

Для использования MKL PARDISO необходимо вызвать функцию **PARDISO** с необходимым набором параметров. Прототип функции приведен ниже.

```
PARDISO (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
         perm, nrhs, iparm, msglvl, b, x, error);
```

Рассмотрим описание каждого из параметров в порядке их следования в прототипе. Для каждого параметра приведем его тип и назначение.

1. **pt**

Массив из 64 указателей на тип **void**.

Рабочая структура данных PARDISO. Для корректной работы необходимо предварительно обнулить все элементы данного массива.

2. **maxfct**

Указатель на **MKL_INT**.

Количество матриц с одним и тем же портретом, которые будут решаться с помощью **PARDISO**.

3. **mnum**

Указатель на **MKL_INT**.

Номер матрицы, которая обрабатывается **PARDISO** в текущем вызове. Нумерация матриц начинается с 1 и заканчивается **maxfct**.

4. **mtype**

Указатель на **MKL_INT**.

Тип матрицы, которая подается на вход:

1 – действительные матрицы с симметричным портретом.

2 – действительные симметричные положительно определенные матрицы.

-2 – действительные симметричные матрицы с неизвестной знакоопределенностью.

3 – комплексные матрицы с симметричным портретом.

4 – Эрмитовы положительно определенные матрицы.

-4 – Эрмитовы матрицы с неизвестной знакоопределенностью.

6 – комплексные симметричные матрицы.

11 – действительные несимметричные матрицы.

13 – комплексные несимметричные матрицы.

5. **phase**

Указатель на **MKL_INT**.

Фаза решения, которую необходимо выполнить в текущем вызове **PARDISO**. Этот параметр может принимать три возможных значения:

0 – освобождение памяти, занимаемой матрицами, хранящими разложение для матрицы с номером **mnum**.

-1 – освобождение памяти, занимаемой всеми матрицами с разложениями.

$10i + j,\ i = \overline{1,3},\ j = \overline{1,3}$, где i – начальная фаза работы, j - финальная фаза работы **PARDISO**. Индексы фаз:

1 – анализ снижения заполнения и численная факторизация.

2 – численная факторизация.

3 – прямой и обратный ход нахождения вектора системы, снижение ошибок вычислений.

$33i, i = \overline{1,3}$ – частичное решение системы. Индексы фаз:

1 – выполняется только прямая подстановка.

2 – выполняется только диагональная подстановка.

3 – выполняется только обратная подстановка.

Допустимые значения параметра **phase**:

11 – анализ снижения заполнения и символическая факторизация.

12 – анализ снижения заполнения, символическая и численная факторизации.

13 – анализ снижения заполнения, символическая и численная факторизации, решение системы и снижение ошибки вычислений.

22 – численная факторизация.

23 – численная факторизация, решение системы и снижение ошибки вычислений.

33 – решение системы и снижение ошибки вычислений.

331 – то же, что 33, выполняется только прямая подстановка.

332 – то же, что 33, выполняется только диагональная подстановка.

333 – то же, что 33, выполняется только обратная подстановка.

6. **n**

Указатель на **MKL_INT**.

Порядок системы.

7. **a, ia, ja**

Матрица в формате CRS.

По умолчанию нумерация массивов **ia** и **ja** начинается с 1, в стиле языка Fortran. Важно отметить, что для симметричных матриц требуется подавать на вход только их верхний треугольник.

8. **perm**

Указатель на массив из элементов типа **MKL_INT** длиной **n**.

Вектор перестановки. В зависимости от дальнейших параметров может быть как входным, так и выходным параметром. Если **perm** играет роль выходного параметра, то в него построчно записывается матрица перестановок, найденная **PARDISO**. Если же **perm** является входным параметром, в таком случае пользователь сам подает матрицу перестановок, которая будет применяться для переупорядочивания. Также данный параметр может не использоваться совсем. Отметим, что, как и все массивы, содержащие индексы, **perm** поддерживает нумерацию с 1.

9. **nrhs**

Указатель на **MKL_INT**.

Количество правых частей системы. При возникновении задачи решения системы со многими правыми частями один раз вызывать **PARDISO** для многих правых частей эффективнее множественного вызова для одной правой части.

10. **iparm**

Указатель на массив из **MKL_INT**, содержащий 64 элемента.

Массив, содержащий либо входные параметры, обеспечивающие настройку **PARDISO**, либо выходную информацию о его работе. Подробное описание параметра **iparm** будет приведено ниже.

11. **msglvl**

Указатель на **MKL_INT**.

Переключатель для вывода статистической информации о ходе решения задачи. Позволяет получить, например, время выполнения каждого этапа (факторизация, переупорядочивание, решение СЛАУ), объем используемой памяти, производительность и пр. Может принимать значения:

0, если информация выводится.

1, если информация не выводится.

Пример вывода статистической информации показан на Рис. 2.

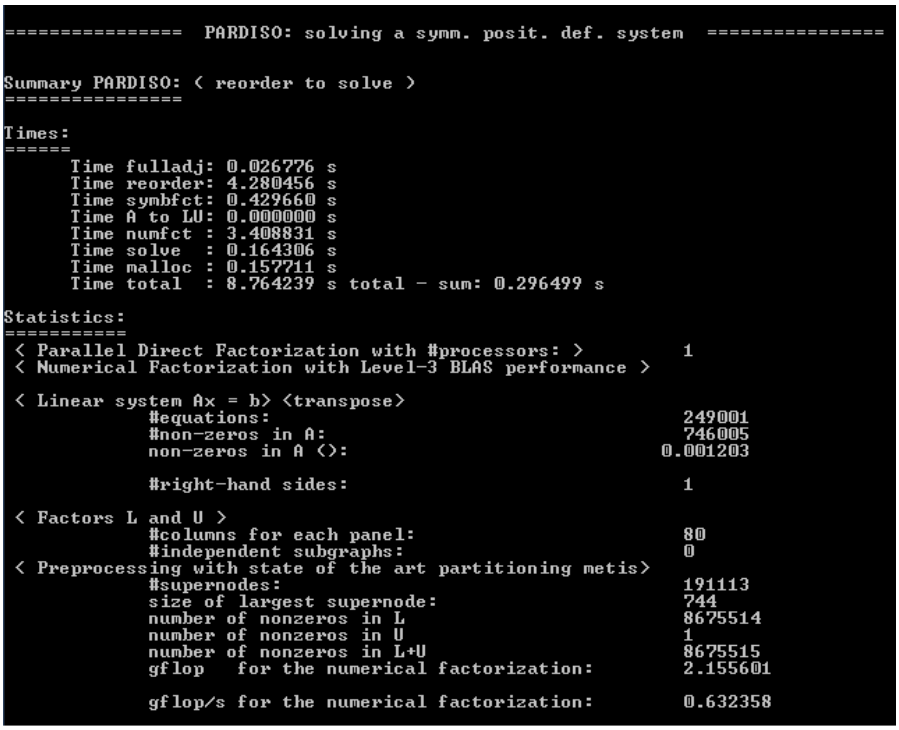


Рис. 2. Пример вывода статистической информации

12. **b**

Указатель на массив из **float**/**double**/**complex** размера **n*nrhs**.

Вектор правых частей. При определенной настройке параметров в этот массив может быть записан искомый вектор системы.

13. **x**

Указатель на массив из **float**/**double**/**complex** размера **n*nrhs**.

Вектор решения. Отметим, что если параметры настроены так, что решение записывается в массив **b**, этот массив все равно необходим как рабочий.

14. **error**

Указатель на **MKL_INT**.

Флаг ошибки. Если значение этого параметра после вызова **PARDISO** равно 0, значит, решатель отработал корректно. Другие значения

параметра **error** свидетельствуют об ошибке. Наиболее часто встречающиеся ошибки:

-1 – ошибка во входных данных.

-2 – недостаточно памяти для работы **PARDISO**.

-3 – проблемы на этапе переупорядочивания.

-4 – произошло деление на нулевой ведущий элемент или ошибка в итерационном уточнении [6].

Полный список возможных ошибок приведен в [9].

15. **iparm**

Массив **iparm** содержит параметры, позволяющие настроить работу **PARDISO**. При обнулении элемента **iparm[0]** массив заполняется значениями по умолчанию. В противном случае **PARDISO** работает согласно установленным значениям. Ниже приведены их численные и семантические значения:

1. **iparm[1]**

Тип переупорядочивания [5]. Доступны следующие типы:

0 – используется метод минимальной степени,

1 – используется метод вложенных сечений (является значением по умолчанию),

2 – используется параллельная OpenMP версия метода вложенных сечений.

2. **iparm[3]**

Параметр настройки итерационного уточнения.

Представляется в виде $10i + j$. Параметр j имеет следующие возможные значения:

0 – решение системы выполняется согласно параметру **phase**.

1 – используется CGS (Conjugate Gradient Squared) предобуславливатель [6] для несимметричных матриц.

2 – факторизация заменяется методом сопряженных градиентов [6] для симметричных матриц.

Параметр i влияет на критерий останова итерационного уточнения, который будет останавливать работу при достижении точности 10^{-i}.

Значение по умолчанию **iparm[3]** равняется 0.

3. **iparm[4]**

Параметр управления матрицей перестановок.

0 – параметр **perm** не используется (значение по умолчанию).

1 – используется пользовательская перестановка.

2 – в массив **perm** записывается перестановка, использованная **PARDISO** для переупорядочивания.

4. **iparm[5]**

Параметр управления вектором решения.

0 – решение записывается в массив **x** (значение по умолчанию).

1 – решение записывается в массив **b**.

5. **iparm[6]**

Количество шагов итерационного уточнения [6]. Выходной параметр, в который записывается количество итераций для уточнения полученного решения.

6. **iparm[9]** , **iparm[10]** , **iparm[12]**

Параметры, которые определяют механизм работы с малыми числами на диагонали.

iparm[9] – величина, которая добавляется к малым числам на диагонали.

iparm[10] – коэффициент, на который умножаются строки/столбцы исходной матрицы таким образом, чтобы на диагонали были единицы.

iparm[12] с помощью несимметричных перестановок добивается возникновения на диагонали больших чисел. Более подробно работа с малыми числами на диагонали описана в [9].

7. **iparm[17]**

Заполнение фактора. Если в этот параметр передано отрицательное значение, то после решения системы в него будет записано количество ненулевых элементов, возникших при разложении матрицы. По умолчанию до вызова **PARDISO** параметр имеет значение -1.

8. **iparm[18]**

Сложность факторизации в MFlops (количество 10^6 операций с плавающей точкой, выполненных за секунду). Если в этот параметр

передано отрицательное значение, то после этапа 11 в него возвращается количество MFlops, выполненных на этапе 22.

9. **iparm[20]**

Параметр выбора ведущего элемента для симметричных матриц с неизвестной знакоопределенностью.

0 – в качестве ведущего выбирается диагональный элемент.

1 – используется Баунч-Кафман [7] обработка диагональных элементов (значение по умолчанию).

10. **iparm[26]**

Маркер проверки корректности входных данных. Отвечает за определение соответствия параметров **ia, ja** формату CRS.

0 – проверка не производится (значение по умолчанию).

1 – проверка производится. В случае некорректных данных (убывание элементов массива **ia**, т.е. не соблюден порядок возрастания столбцов в строке) **PARDISO** заканчивает работу с кодом ошибки -1.

Отметим, что проверка корректности входных данных может сильно замедлить работу решателя.

11. **iparm[27]**

Параметр, определяющий тип данных с плавающей точкой.

0 – входные параметры **a**, **x**, **b** имеют тип **double** (значение по умолчанию).

1 – входные параметры **a**, **x**, **b** имеют тип **float**.

12. **iparm[59]**

Маркер управления режимом Out-Of-Core (OOC). Режим OOC решает проблемы хранения слишком большого фактора матрицы путем его записи в файл на диск в отличие от режима In-Core, когда вся информация хранится в оперативной памяти.

0 – используется режим In-Core (значение по умолчанию).

1 – используется режим In-Core, если общий объем памяти (в Mb), требуемый для хранения фактора матрицы, меньше значения переменной окружения **MKL_PARDISO_OOC_MAX_CORE_SIZE** (ее значение по умолчание 2000Mb), и режим OOC в противном случае.

2 – используется OOC режим.

Остальные параметры зарезервированы для будущего использования и должны быть выставлены в ноль.

2.4. Выделение этапа переупорядочивания матрицы с использованием PARDISO. Пример вызова

Описав параметры вызова решателя MKL PARDISO, перейдем к рассмотрению примеров выделения некоторых фаз решения СЛАУ с помощью данного решателя. В настоящем разделе приведем пример выделения этапа переупорядочивания матрицы.

Вспомним, что для переупорядочивания матрицы и оценки заполнения ее фактора необходимо вызвать PARDISO с параметром **phase**, равным 11. Результатом работы могут быть количество ненулевых элементов (параметр **iparm[17]**) в факторе или матрица перестановок (параметр **perm**), использованная для переупорядочивания. Пример вызова **PARDISO** для переупорядочивания матрицы методом вложенных сечений приведен ниже. Будем считать, что задана симметричная положительно определенная матрица с действительными элементами в формате CRS. Программная структура для хранения разреженных матриц в формате CRS была описана в разделе 2.2.

```
int maxfct = 1,
    mnum = 1,
    mtype = 2, // матрица симметрична и положительно
               // определена
    phase = 11, // переупорядочивание и символическая
                // факторизация
    *perm = new int[n], // перестановка
    nrhs = 1, // число правых частей
    msglvl = 1, // вывод статистической информации
    error = 0; // код ошибки

int iparm[64];
iparm[0] = 0; // используются параметры по умолчанию

double *b = NULL;
double *x = NULL;

PARDISO (pt, &maxfct, &mnum, &mtype, &phase, &n, value,
rowIndex, col, perm, &nrhs, iparm, &msglvl, b, x, &error);

printf("Number of nonzeros in U: %d\n", iparm[17]);
```

Поскольку первый элемент массива **iparm** равен 0, то все остальные его элементы заполняются значениями по умолчанию, в частности, **iparm[1]**, отвечающий за выбор способа переупорядочивания, будет равен 2, что соответствует методу вложенных сечений. Для выбора другого

способа переупорядочивания необходимо инициализировать ненулевым числом **iparm[0]** и записать в **iparm[1]** значение, соответствующее идентификатору выбранного метода переупорядочивания. Ниже приведен пример заполнения массива **iparm** для выбора метода минимальной степени.

```
int iparm[64];
iparm[0] = 1; // заполнение массива параметров
// пользователем
iparm[1] = 0; // используется метод минимальной степени
iparm[17] = -1; // записать в эту переменную количество
// ненулевых значений в матрице U
```

Аналогично можно использовать параллельную реализацию метода вложенных сечений.

Вызов фазы переупорядочивания позволяет оценить пользовательскую перестановку с точки зрения заполнения фактора. В этом случае нужно определить матрицу перестановок через параметр **perm** и с помощью параметра **iparm[4]** использовать ее для переупорядочивания. Пример символической факторизации матрицы без переупорядочивания приведен ниже.

```
int maxfct = 1,
    mnum = 1,
    mtype = 2,
    phase = 11,
    *perm = new int[n],
    nrhs = 1,
    msglvl = 1,
    error = 0;

int iparm[64];
iparm[0] = 1;
iparm[4] = 1; // используется пользовательская перестановка
iparm[17] = -1;

// формируем прямую перестановку
for(int i = 0; i < n; i++)
    perm[i] = i + 1;

double *b = NULL;
double *x = NULL;

PARDISO (pt, &maxfct, &mnum, &mtype, &phase, &n, value,
rowIndex, col, perm, &nrhs, iparm, &msglvl, b, x, &error);

printf("Number of nonzeros in U: %d\n", iparm[17]);
```

2.5. Факторизация матрицы и решение системы с факторизованной матрицей с помощью PARDISO. Пример вызова

Для решения исходной системы необходимо пройти все фазы решателя: переупорядочивание, факторизация и решение вспомогательных систем. Так же, как и в предыдущем разделе будем считать, что матрица и вектор правых частей определены. Тогда вызов PARDISO может иметь следующий вид:

```
int maxfct = 1,
    mnum = 1,
    mtype = 2,
    phase = 13, // факторизация и решение системы
    *perm = new int[n],
    nrhs = 1,
    msglvl = 1,
    error = 0;

int iparm[64];
iparm[0] = 0;

double *x = new double[n];

PARDISO (pt, &maxfct, &mnum, &mtype, &phase, &n, value,
rowIndex, col, perm, &nrhs, iparm, &msglvl, b, x, &error);
```

Такой вызов обеспечивает решение системы с одной правой частью и сохранением вектора решения в массив **x**.

2.6. Дополнительный способ вызова (DSS интерфейс)

Разработчики MKL PARDISO предоставляют дополнительный способ вызова через Direct Sparse Solver (DSS) интерфейс. DSS интерфейс состоит из набора функций, соответствующих основным фазам решателя. В отличие от основного интерфейса эти функции имеют упрощенный набор параметров. Подробное описание приведено в документации [9], здесь же дадим краткое описание его некоторых возможностей. Работа решателя инициализируется вызовом функции **dss_create**.

```
dss_create(handle, opt);
```

Эта функция создает необходимые для работы решателя структуры данных. Параметр **opt** является обязательным для всех функций набора DSS интерфейса и позволяет делать некоторые настройки работы PARDISO. Параметр **opt** в функции **dss_create** определяет количество шагов итерационного уточнения, точность вещественных чисел и параметры OOC режима (подробнее п. 12, глава 2.3).

На втором этапе определяется структура разреженности матрицы с помощью функции:

```
dss_define_structure(handle, opt, rowIndex, nRows, nCols,
                     columns, nNonZeros);
```

Ей передается портрет матрицы в формате CRS и с помощью **opt** указывается структурный тип матрицы.

Дальше выполняется переупорядочивание и символическая факторизация:

```
dss_reorder(handle, opt, perm);
```

Здесь возможно использовать переупорядочивание по умолчанию, параллельную версию метода вложенных сечений или пользовательскую перестановку.

Численные значения фактора вычисляются с помощью одной из функций, приведенных ниже, в зависимости от типа элементов матрицы (действительного или комплексного).

```
dss_factor_real(handle, opt, rValues);

dss_factor_complex(handle, opt, cValues);
```

Параметр **opt** в данном случае контролирует знакоопределенность матрицы.

Аналогичная пара функций применяется для решения системы.

```
dss_solve_real(handle, opt, rRhsValues, nRhs, rSolValues);

dss_solve_complex(handle, opt, cRhsValues, nRhs,
                  cSolValues);
```

Параметр **opt** при решении СЛАУ позволяет вычислять искомый вектор системы либо с помощью решения всех вспомогательных систем аналогично значению **phase=33** при прямом вызове **PARDISO**, либо только при помощи одной из подстановок, что соответствует **phase = 331**, **phase = 332** или **phase = 333**.

Статистическую информацию о работе каждой из фаз можно получить, поставив после соответствующей функции вызов функции:

```
dss_statistics(handle, opt, statArr, retValues);
```

Данная функция записывает в массив **retValues** время выполнения и затраченное количество памяти.

Завершение работы решателя и освобождение созданных в процессе работы структур данных происходит с помощью функции **dss_delete**.

```
dss_delete(handle, opt);
```

В целом, спектры возможностей интерфейса DSS и прямого вызова **PARDISO** совпадают, но последний предполагает более тщательную настройку параметров, в частности, переупорядочивания и итерационного уточнения. Таким образом, выбор способа работы с MKL PARDISO обусловлен пользовательскими требованиями к методике решения.

3. Некоторые возможности прямого решателя СЛАУ в составе Intel® Math Kernel Library LAPACK

Задача вычисления распределения температур в прямоугольной пластине в конечном счете сводится к решению СЛАУ с ленточной матрице, поэтому для контроля корректности имеет смысл сравниваться со специализированным решателем.

Наряду с пакетом MKL PARDISO разработчики библиотеки MKL предоставляют возможность решать системы с ленточной матрицей с помощью специально предназначенных решателей. В их основе также лежит разложение матрицы и решение треугольных систем, адаптированные под ленточную структуру разреженности. Для симметричных и положительно определенных матриц существует функция **dpbtrf** для поиска разложения, которая в дальнейшем используется решателем **dpbtrs**. Указанные функции входят в состав пакета MKL LAPACK.

3.1. Типы поддерживаемых матриц, способ хранения ленточных матриц

Функции решения СЛАУ с ленточными матрицами работают со специальным форматом хранения. В общем случае хранятся все диагонали ленты, начиная с верхней. Пусть для матрицы размера $n \times n$ k_l – число диагоналей под главной диагональю, k_u – количество диагоналей над главной диагональю. Тогда для ленточного хранения требуется двумерный массив размером $n \times (k_l + k_u + 1)$, где элемент с индексами $i, j, i = \overline{1, n}, j = \overline{1, k_l + k_u + 1}$ означает элемент -й диагонали, находящийся в i-м столбце. Отметим, что диагонали нумеруются, начиная с верхней, а главная диагональ имеет номер $k_l + 1$.

Пусть матрица размером 5×5 имеет вид:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ a_{21} & a_{22} & a_{23} & a_{24} & \\ & a_{32} & a_{33} & a_{34} & a_{35} \\ & & a_{43} & a_{44} & a_{45} \\ & & & a_{54} & a_{55} \end{pmatrix}.$$

Тогда $k_l = 1, k_u = 2$, а ее запись в ленточном формате такова:

$$\begin{pmatrix} * & * & a_{11} & * \\ * & a_{12} & a_{22} & a_{21} \\ a_{13} & a_{23} & a_{33} & a_{32} \\ a_{24} & a_{34} & a_{44} & a_{43} \\ a_{35} & a_{45} & a_{55} & a_{54} \end{pmatrix}$$

Элементы, отмеченные *, не используются и могут принимать любое значение.

В системе, возникающей при решении задачи теплопроводности, матрица симметрична, поэтому достаточно хранить половину ее элементов, например, верхний треугольник, и главную диагональ. Для ленточной формы записи симметричной матрицы используется массив $n \times (k_l + 1)$.

Пусть матрица размером 5×5 имеет вид:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ a_{12} & a_{22} & a_{23} & a_{24} & \\ a_{13} & a_{23} & a_{33} & a_{34} & a_{35} \\ & a_{24} & a_{34} & a_{44} & a_{45} \\ & & a_{35} & a_{45} & a_{55} \end{pmatrix}.$$

Ей соответствует массив:

$$\begin{pmatrix} * & * & a_{11} \\ * & a_{12} & a_{22} \\ a_{13} & a_{23} & a_{33} \\ a_{24} & a_{34} & a_{44} \\ a_{35} & a_{45} & a_{55} \end{pmatrix}.$$

3.2. Параметры вызова функции dpbtrs

Перед непосредственным использованием решателя **dpbtrs** необходимо сначала выполнить вызов функции **dpbtrf**, которая находит разложение матрицы. Приведем ее прототип и описание параметров.

```
dpbtrf(uplo, n, kd, ab, ldab, info);
```

1. **uplo**

Указатель на символьный тип, **char***.

Значение по указателю должно быть ‘U’ или ‘L’. В зависимости от значения **uplo** подаваемая на вход матрица рассматривается либо как верхнетреугольная (**uplo = 'U'**), либо как нижнетреугольная (**uplo = 'L'**).

2. **n**

Указатель на **int**.

Порядок матрицы.

3. **kd**

Указатель на **int**.

Количество верхних или нижних диагоналей матрицы.

4. **ab**

Указатель на массив элементов типа **double**.

Содержит исходную матрицу в ленточной симметричной записи. Между значением параметра **uplo** и выбором хранения **ab** должно быть строгое соответствие. После вызова функции в этот массив записывается разложение матрицы.

5. **ldab**

Указатель на **int**.

Содержит значение первой размерности массива **ab.** В случае симметричной матрицы этот параметр равен **kd+1.**

6. **info**

Указатель на **int**.

Выходной параметр, в который записывается код ошибки работы функции. Если после вызова функции он равен 0, то это означает, что факторизация выполнилась корректно. Если **info = -i**, то **i**-й параметр принимает недопустимое значение. Если **info = i**, то главный минор **i**-го порядка исходной матрицы неположительно определен, и факторизация не может быть завершена.

После факторизации матрицы вызывается функция **dpbtrs** для решения системы:

```
dpbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info);
```

Часть ее параметров повторяет одноименные параметры функции **dpbtrf**, поэтому приведем только описание некоторых из них.

1. **nrhs**

Указатель на **int**.

Количество правых частей в системе.

2. **ab**

Указатель на массив из **double**. Содержит факторизованную матрицу.

3. **b**

Указатель на двумерный массив из **double** размером **ldb** × **nrhs**. Содержит **nrhs** векторов правых частей.

4. **ldb**

Указатель на **int**.

Первая размерность массива **b**. Это значение совпадает с порядком матрицы, то есть с **n**.

4. Стационарная задача вычисления распределения температур в пластине

Пусть l_1, l_2 – длины сторон прямоугольной пластины, $U(x,y)$ – температура пластины в точке (x,y), принадлежащей области G, $G = \{(x,y): x \in [0,l_1], y \in [0,l_2]\}$, а $f(x,y)$ – суммарное воздействие внешних источников и стоков на прямоугольную пластину в точке (x,y). Стационарный процесс распределения температур в пластине описывается дифференциальным уравнением Пуассона:

$$\Delta U = U_{xx} + U_{yy} = -f(x,y)$$
$$G = \{(x,y): x \in [0,l_1], y \in [0,l_2]\} \quad (1)$$

Для полного описания стационарного процесса необходимо задать температурный режим на границе пластины:

$$U(0,y) = \mu_1(y) \quad (2)$$
$$U(l_1,y) = \mu_2(y)$$
$$U(x,0) = \mu_3(x)$$
$$U(x,l_2) = \mu_4(x)$$

Отметим, что в угловых точках пластины должны выполняться условия согласованности граничных условий:

$$U_{0,0} = \mu_1(0) = \mu_3(0),$$
$$U_{0,l_2} = \mu_1(l_2) = \mu_4(0),$$
$$U_{l_1,0} = \mu_2(0) = \mu_3(l_1),$$
$$U_{l_1,l_2} = \mu_2(l_2) = \mu_4(l_1).$$

Для определенности в качестве функций $f(x,y), \mu_1(y), \mu_2(y), \mu_3(x), \mu_4(x)$ возьмем следующие функции:

$$f(x,y) = -2(x+y), \quad (3)$$

$$\mu_1(y) = 0, \quad (4)$$

$$\mu_2(y) = l_1 y(l_1 + y), \quad (5)$$

$$\mu_3(x) = 0, \quad (6)$$

$$\mu_4(x) = l_2 x(l_2 + x). \quad (7)$$

Такой выбор функций соответствует уравнению теплопроводности с точным решением в виде:

$$U(x,y) = xy(x+y). \quad (8)$$

5. Вычислительная схема

Введем сетку $x_i = ih, y_j = jk$, где $h = \frac{l_1}{n}, k = \frac{l_2}{m}$ на области G. Для аппроксимации дифференциального уравнения (1) с граничными условиями (2) будем использовать крестообразный шаблон. Аппроксимируем дифференциальное уравнение разностной схемой (9).

$$\frac{(U_{i-1,j} - 2U_{i,j} + U_{i+1,j})}{h^2} + \frac{(U_{i,j-1} - 2U_{i,j} + U_{i,j+1})}{k^2} = -f_{i,j},\ i = \overline{1, n-1}, j = \overline{1, m-1}$$

$$U_{0,j} = \mu_{1,j},\ j = \overline{0,m} \quad (9)$$

$$U_{n,j} = \mu_{2,j},\ j = \overline{0,m}$$

$$U_{i,0} = \mu_{3,i},\ i = \overline{0,n}$$

$$U_{i,m} = \mu_{4,i},\ i = \overline{0,n}$$

Граничные условия разрешены относительно неизвестных в граничных узлах сетки. Поэтому уравнения, содержащие граничные условия, можно убрать из системы разностных уравнений, подставив вместо соответствующих неизвестных их значения. Рассмотрим первое уравнение разностной схемы ($i = 1$):

$$\frac{(U_{0,j} - 2U_{1,j} + U_{2,j})}{h^2} + \frac{(U_{1,j-1} - 2U_{1,j} + U_{1,j+1})}{k^2} = -f_{i,j},\ j = \overline{2, m-2}$$

$$\frac{U_{2,j}}{h^2} - 2\left(\frac{1}{h^2} + \frac{1}{k^2}\right)U_{1,j} + \frac{U_{1,j-1}}{k^2} + \frac{U_{1,j+1}}{k^2} = -f_{i,j} - \frac{1}{h^2}\mu_{1,j},\ j = \overline{2, m-2}$$

$$\frac{(U_{0,1} - 2U_{1,1} + U_{2,1})}{h^2} + \frac{(U_{1,0} - 2U_{1,1} + U_{1,2})}{k^2} = -f_{i,1},\ j = 1$$

$$\frac{U_{2,1}}{h^2} - 2\left(\frac{1}{h^2} + \frac{1}{k^2}\right)U_{1,1} + \frac{U_{1,2}}{k^2} = -f_{i,1} - \frac{1}{k^2}\mu_{3,1} - \frac{1}{h^2}\mu_{1,1},\ j = 1$$

$$\frac{(U_{0,m-1} - 2U_{1,m-1} + U_{2,m-1})}{h^2} + \frac{(U_{1,m-2} - 2U_{1,m-1} + U_{1,m})}{k^2} = -f_{i,j},\ j = m-1$$

$$\frac{U_{2,m-1}}{h^2} - 2\left(\frac{1}{h^2} + \frac{1}{k^2}\right)U_{1,m-1} + \frac{U_{1,m-2}}{k^2} =$$
$$-f_{i,j} - \frac{1}{h^2}\mu_{1,m-1} - \frac{1}{k^2}\mu_{4,1}, \qquad j = m-1$$

Аналогично можно рассмотреть случаи, когда $i = n-1$ и $i = \overline{2, n-2}$.

Сформируем вектор неизвестных $\vec{U}$ следующим образом: $\vec{U} = (U_{1,1}, U_{2,1}, \dots, U_{n-1,1}, U_{1,2}, U_{2,2}, \dots, U_{n-1,2}, \dots, U_{1,m-1}, U_{2,m-1}, \dots, U_{n-1,m-1})$ - внутренние узлы сетки, которые перечислены в порядке, соответствующем их расположению вдоль оси изменения координаты x.

Тогда систему разностных уравнений можно записать в матричном виде $A\vec{U} = \vec{F}$, где $\vec{F}$ – это вектор значений функции $f(x, y)$ во внутренних узлах сетки минус «довески» от граничных условий, а матрица A – блочная пятидиагональная матрица, причем каждая диагональ в отдельности содержит одинаковое число. На Рис. 3 показан пример такой матрицы для случая $n = 5, m = 5$ (где коэффициент $A = -2\left(\frac{1}{h^2} + \frac{1}{k^2}\right)$).

A	$\frac{1}{h^2}$	0	0	$\frac{1}{k^2}$	0	0	0								
$\frac{1}{h^2}$	A	$\frac{1}{h^2}$	0	0	$\frac{1}{k^2}$	0	0								
0	$\frac{1}{h^2}$	A	$\frac{1}{h^2}$	0	0	$\frac{1}{k^2}$	0								
0	0	$\frac{1}{h^2}$	A	0	0	0	$\frac{1}{k^2}$								
$\frac{1}{k^2}$				A	$\frac{1}{h^2}$			$\frac{1}{k^2}$							
	$\frac{1}{k^2}$			$\frac{1}{h^2}$	A	$\frac{1}{h^2}$			$\frac{1}{k^2}$						
		$\frac{1}{k^2}$			$\frac{1}{h^2}$	A	$\frac{1}{h^2}$			$\frac{1}{k^2}$					
			$\frac{1}{k^2}$			$\frac{1}{h^2}$	A				$\frac{1}{k^2}$				
				$\frac{1}{k^2}$				A	$\frac{1}{h^2}$			$\frac{1}{k^2}$			
					$\frac{1}{k^2}$			$\frac{1}{h^2}$	A	$\frac{1}{h^2}$			$\frac{1}{k^2}$		
						$\frac{1}{k^2}$			$\frac{1}{h^2}$	A	$\frac{1}{h^2}$			$\frac{1}{k^2}$	
							$\frac{1}{k^2}$			$\frac{1}{h^2}$	A				$\frac{1}{k^2}$

								$\frac{1}{k^2}$				A	$\frac{1}{h^2}$		
									$\frac{1}{k^2}$			$\frac{1}{h^2}$	A	$\frac{1}{h^2}$	
										$\frac{1}{k^2}$			$\frac{1}{h^2}$	A	$\frac{1}{h^2}$
											$\frac{1}{k^2}$			$\frac{1}{h^2}$	A

Рис. 3. Пример матрицы A для сетки размерности $n = 5, m = 5$

Полученная матрица будет отрицательно определенной, поэтому для дальнейшего применения метода Холецкого для решения положительно определенных систем будем решать эквивалентную систему $-A\vec{U} = -\vec{F}$.

6. Программная реализация

Перейдем к программной реализации решения уравнения теплопроводности с помощью MKL PARDISO. Для этого необходимо последовательно выполнить следующие шаги:

- Запустите приложение **Microsoft Visual Studio 2008**.
- В меню **File** выполните команду **New→Project…**.
- В диалоговом окне **New Project** в типах проекта выберите **Win32**, в шаблонах **Win32 Console Application**, в поле **Solution** введите **HCE**, в поле **Name** – **HCE_PARDISO**, в поле **Location** укажите путь к папке с лабораторными работами курса – **c:\ParallelCalculus**. Нажмите **OK**.
- В диалоговом окне **Win32 Application Wizard** нажмите **Next** (или выберите **Application Settings** в дереве слева) и установите флаг **Empty Project**. Нажмите **Finish**.
- В окне **Solution Explorer** в папке **Source Files** выполните команду контекстного меню **Add→New Item…**. В дереве категорий слева выберите **Code**, в шаблонах справа – **C++ File (.cpp)**, в поле **Name** введите имя файла **main**. В этом файле будут располагаться вызовы основных функций. Нажмите **Add**.
- В окне **Solution Explorer** в папке **Header Files** выполните команду контекстного меню **Add→New Item…**. В дереве категорий слева выберите **Code**, в шаблонах справа – **Header File (.h)**, в поле **Name** введите имя файла **PardisoCall**. Нажмите **Add**. В данном файле

будет находиться прототип функции, вызывающей MKL PARDISO и подготавливающей необходимые для его работы параметры.

- Аналогично создайте файл **PardisoCall.cpp**, где будет размещаться реализация вызова PARDISO, а также файлы **Utilities.h** и **Utilities.cpp** для вспомогательных функций, файлы **PoissonDecision.h** и **PoissonDecision.cpp** для функций, задающих правую часть, граничные условия и решение дифференциального уравнения, и файлы **DpbtrsCall.h** и **DpbtrsCall.cpp** для функции вызова ленточного решателя.

- Чтобы собрать 32-битное приложение с вызовом приведенных функций MKL, нужно прописать пути до заголовочных и .lib файлов библиотеки. Для этого в меню **Tools** выполните команду **Options**. В появившемся диалоговом окне выберите вкладку **Projects and Solutions→VC++ Directories**. В выпадающем меню выберите сначала **Include Files** и добавьте новую запись с путем к заголовочным файлам библиотеки MKL (к примеру, **C:\Program Files (x86)\Intel\ComposerXE-2011\mkl\include**), затем выберите **Library Files** и добавьте путь к библиотечным файлам для платформы ia-32 (к примеру, **C:\Program Files (x86)\Intel\ComposerXE-2011\mkl\lib\ia32**). Для сборки 64-битного приложения необходимо указать путь к статическим библиотекам для 64-битной платформы (к примеру, **C:\Program Files (x86)\Intel\ComposerXE-2011\mkl\lib\intel64**).

- Далее нужно выбрать конкретные статические библиотеки, в которых находятся реализации используемых функций. Для 32-битного приложения это файлы **mkl_core.lib**, **mkl_intel_c.lib**, **mkl_sequential.lib**, **mkl_solver_sequential.lib**, для 64-битного: **mkl_core.lib**, **mkl_sequential.lib**, **mkl_intel_lp64.lib**, **mkl_blas95_lp64.lib**. Для их подключения во вкладке **Linker→Input→Additional Dependencies** в свойствах проекта **Configuration Properties** необходимо вписать требуемый набор библиотек.

Перейдем к реализации. В файле **main.cpp** создадим заготовку функции **`main()`**. Через параметры командной строки передадим размер сетки n, m.

```
#include "PardisoCall.h"
#include "DpbtrsCall.h"

int main(int argc, char* argv[])
{
  if(argc != 3)
  {
    printf("Invalid input parameters.\n HCE_PARDISO n
      m\n \t(n,m) - grid sizes.\n");
```

```
    return 1;
  }

  int n = atoi(argv[1]);
  int m = atoi(argv[2]);

  int size = (n - 1) * (m - 1);
  double * u = new double[size];

  double time_pds = computeDecision(n, m, u);
  double err_pds = CalculateError(n, m, u);

  double time_band = computeDecision_Band(n, m, u);
  double err_band = CalculateError(n, m, u);

  // печать выходной информации:
  // (n,m) - размеры сетки,
  // time_pds - время работы MKL PARDISO,
  // err_pds - невязка решения, найденного MKL PARDISO,
  // time_band - время работы ленточного решателя MKL,
  // err_band - невязка решения, найденного ленточным
  //            решателем MKL.
  printf("%d;%d;%.15f;%.15f;%.15f;%.15f\n", n, m, time_pds,
    err pds, time band, err band);

  delete[] u;
  return 0;
}
```

В случае, когда количество входных параметров отличается от трех, например, в функцию main не передаются размеры сетки, выводится сообщение об ошибке следующего вида:

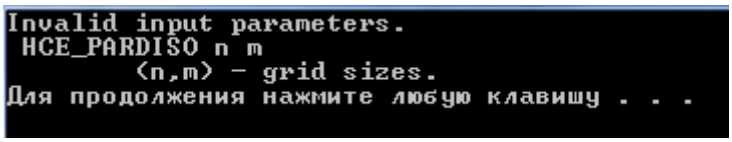


Рис. 4. Сообщение об ошибке ввода неправильного количества параметров

В случае получения корректных входных параметров получаем размеры сетки с помощью стандартной функции **`atoi`** и выделяем память под искомый вектор системы. Функции **`computeDecision`** и **`computeDecision_Band`** получают решение системы с помощью MKL PARDISO и ленточного решателя **`dpbtrs`** соответственно. Функция **`CalculateError`** вычисляет модуль максимального отклонения найденных значений решения СЛАУ от искомой функции дифференциального уравнения. Отметим, что задача построена таким образом, что точное решение дифференциального уравнения известно. По

завершении функции **main** пользователь получает информацию о времени работы и погрешности каждого из решателей.

6.1. Вспомогательные функции

В данном разделе рассмотрим функции, формирующие матрицы и вектор правой части. Для перехода от решения дифференциального уравнения к решению СЛАУ используем функции, задающие правую часть, граничные условия и точное решение дифференциального уравнения, с помощью которого будем проверять корректность решения задачи. Прототипы и реализации функций будут располагаться в файлах **PoissonDecision.h** и **PoissonDecision.cpp**. Ниже приведен фрагмент файла **PoissonDecision.cpp**, где реализованы функции, определяющие рассматриваемую граничную задачу С помощью констант **L1** и **L2** задаются размеры пластины. Функции реализованы согласно формулам (3)-(8) главе 4.

```
#define L1 10.0
#define L2 10.0

// Uxx + Uyy = -f(x,y)
// Функция вычисления правой части ДУ в ЧП
double f(double x, double y)
{
  return -2.0 * (x + y);
}

// Функция вычисления значения ГУ на левой стороне
прямоугольника
double mu1(double y)
{
  return 0;
}

// Функция вычисления значения ГУ на правой стороне
//прямоугольника
double mu2(double y)
{
  return L1 * y * (L1 + y);
}

// Функция вычисления значения ГУ на нижней стороне
//прямоугольника
double mu3(double x)
{
  return 0;
}

// Функция вычисления значения ГУ на верхней стороне
//прямоугольника
```

```
double mu4(double x)
{
  return L2 * x * (x + L2);
}

// Функция, задающая точное решение дифференциального
// уравнения
double u_(double x, double y)
{ return x * x * y + y * y * x; }
```

Функция расчета погрешности, а также функции выделения и освобождения памяти, занимаемой матрицей в формате CRS, располагаются в файлах **Utilities.h** и **Utilities.cpp**. Рассмотрим каждую из этих функций подробнее.

Инициализацию и удаление разреженной матрицы будет выполнять пара функций **`InitializeMatrix`** и **`FreeMatrix`**.

```
void InitializeMatrix(int N, int NZ, crsMatrix &mtx)
{
  mtx.N = N;
  mtx.NZ = NZ;
  mtx.Value    = new double[NZ];
  mtx.Col      = new int[NZ];
  mtx.RowIndex = new int[N + 1];
}

void FreeMatrix(crsMatrix &mtx)
{
  delete [] mtx.Value;
  delete [] mtx.Col;
  delete [] mtx.RowIndex;
}
```

Для расчета погрешности решения дифференциального уравнения создадим функцию **`CalculateError`**. Она будет принимать размеры сетки и решение системы в виде массива элементов типа **`double`** и возвращать модуль максимального отклонения значения приближенной функции от значения точной функции **`u_`** по всем узлам сетки:

```
double CalculateError(int n, int m, double * u)
{
  double h = L1 / n, k = L2 / m;

  double max_err = 0.0;
  for(int i = 0; i < m - 1; i++)
    for(int j = 0; j < n - 1; j++)
    {
      double exact = u_(double(j + 1) * h,
                        double(i + 1) * k);
      double approx = u[i * (n - 1) + j];
```

```
      double tmp_err = fabs(exact - approx);
      if(tmp_err > max_err)
        max_err = tmp_err;
    }

  return max_err;
}
```

6.2. Последовательная версия №1. Реализация функции решения СЛАУ с использованием последовательного MKL PARDISO

Теперь рассмотрим функции, инициализирующие значения элементов матрицы и вектора правых частей. Назовем функцию, формирующую матрицу в формате CRS, **`InitializeHCEMatrixCRS`**, а вектор правых частей – **`InitializeRHVector`**. Ниже приведен код функции **`InitializeHCEMatrixCRS`**.

```
void InitializeHCEMatrixCRS(int n, int m, crsMatrix &A)
{
  double h = L1 / n;
  double k = L2 / m;

  InitializeMatrix((n - 1) * (m - 1), 3 * m * n - 4 * n - 4
    * m + 5, A);

  double inv_h2 = 1.0 / (h * h);
  double inv_k2 = 1.0 / (k * k);
  double diag = 2 * (inv_h2 + inv_k2);

  int i_a = 0;
  for(int i = 0; i < m - 2; i++)
  {
    for(int j = 0; j < n - 2; j++)
    {
      A.Value[i_a] = diag;
      A.Col[i_a] = i * (n - 1) + j;

      A.Value[i_a + 1] = - inv_h2;
      A.Col[i_a + 1] = i * (n - 1) + j + 1;

      A.Value[i_a + 2] = - inv_k2;
      A.Col[i_a + 2] = i * (n - 1) + j + n - 1;

      i_a += 3;
    }

    A.Value[i_a] = diag;
    A.Col[i_a] = i * (n - 1) + n - 2;
```

```
    A.Value[i_a + 1] = - inv_k2;
    A.Col[i_a + 1] = i * (n - 1) + 2 * n - 3;

    i_a += 2;
  }

  int shift = (n - 1) * (m - 2);
  for(int i = 0; i < n - 2; i++)
  {
    A.Value[i_a] = diag;
    A.Col[i_a] = shift + i;

    A.Value[i_a + 1] = - inv_h2;
    A.Col[i_a + 1] = shift + i + 1;

    i_a += 2;
  }

  A.Value[i_a] = diag;
  A.Col[i_a] = A.N - 1;

  A.RowIndex[0] = 0;
  for(int i = 0; i < shift; i++)
  {
    if(i % (n - 1) < n - 2)
      A.RowIndex[i + 1] = A.RowIndex[i] + 3;
    else
      A.RowIndex[i + 1] = A.RowIndex[i] + 2;
  }

  for(int i = 0; i < n - 2; i++)
    A.RowIndex[shift + 1 + i] = A.RowIndex[shift + i] + 2;

  A.RowIndex[A.N] = A.RowIndex[A.N - 1] + 1;
}
```

Для вызова PARDISO необходимо сформировать матрицу в формате CRS, причем в силу симметричности необходимо хранить только ее верхний треугольник. Глядя на структуру матрицы, приведенную в главе 5, можно определить, что ее порядок равен $(n-1)\times(m-1)$, а количество ненулевых элементов в верхнетреугольном представлении совпадает с $3mn-4(m+n)+5$. Предлагаем читателю самостоятельно в этом убедиться. Зная эти параметры матрицы, можно выделить память, необходимую для хранения матрицы в формате CRS, с помощью функции **`InitializeMatrix`**. Следует отметить, что ненулевые элементы в матрице могут принимать только три значения: $2\left(\frac{1}{h^2}+\frac{1}{k^2}\right)$, $-\frac{1}{h^2}$ и $-\frac{1}{k^2}$. Соответственно обозначим в коде программы их как **`diag`**, **`inv_h2`** и

inv_k2. Можно отметить закономерность в расположении ненулевых элементов матрицы, которая позволит сконструировать цикл, заполняющий массивы **Value**, **Col** и **RowIndex**. Она заключается в том, что построчно матрицу можно разделить на $m-1$ блоков шириной в $n-1$ строку. При этом первые $m-2$ блока подчиняются общему правилу: первые $n-2$ строки блока содержат по 3 ненулевых элемента $2\left(\frac{1}{h^2}+\frac{1}{k^2}\right)$, $-\frac{1}{h^2}$ и $-\frac{1}{k^2}$, а последняя строка содержит 2 ненулевых элемента $2\left(\frac{1}{h^2}+\frac{1}{k^2}\right)$ и $-\frac{1}{k^2}$. Строки последнего блока содержат по 2 два ненулевых элемента $2\left(\frac{1}{h^2}+\frac{1}{k^2}\right)$ и $-\frac{1}{h^2}$, за исключением последней строки, в которой есть только диагональный элемент. Двойной цикл формирует $m-2$ блока, после чего отдельный цикл заполняет строки последнего блока.

Теперь перейдем к функции **InitializeRHVector**, которая отвечает за вычисление значений вектора правых частей, которые являются комбинацией значений функции **f** в правой части дифференциального уравнения и функций граничных условий. Если расписать формулу (8), то получится, что каждая компонента вектора включает значение функции **f**, каждая первая компонента в блоке включает довесок от функции **mu1**, а каждая последняя – от функции **mu2**. При этом все компоненты первого блока включают значения функции **mu3**, а все компоненты последнего – функции **mu4**. Приведенный ниже код иллюстрирует выполненные рассуждения.

```
void InitializeRHVector(int n, int m, double * & b)
{
  b = new double[(n - 1) * (m - 1)];

  double h = L1 / n, k = L2 / m;
  double inv_h2 = 1.0 / (h * h);
  double inv_k2 = 1.0 / (k * k);

  for(int i = 0; i < m - 1; i++)
    for(int j = 0; j < n - 1; j++)
      b[i * (n - 1) + j] = f(double(j + 1) * h, double(i +
        1) * k);

  // добавление граничных условий v_0j - mu1
  for(int i = 0; i < m - 1; i++)
    b[i * (n - 1)] += mu1(double(i + 1) * k) * inv_h2;

  // добавление граничных условий v_i0 - mu3
  for(int i = 0; i < n - 1; i++)
    b[i] += mu3(double(i + 1) * h) * inv_k2;

  // добавление граничных условий v_nj - mu2
  for(int i = 0; i < m - 1; i++)
```

```
    b[i * (n - 1) + n - 2] += mu2(double(i + 1) * k) *
inv_h2;

  // добавление граничных условий v_im - mu4
  for(int i = 0; i < n - 1; i++)
    b[(m - 2) * (n - 1) + i] += mu4(double(i + 1) * h) *
inv_k2;
}
```

После реализации вспомогательных функций можно перейти непосредственно к работе с MKL PARDISO. Для его использования в файле **PardisoCall.h** необходимо подключить файлы **mkl_pardiso.h** и **Utilities.h** и описать прототип функции, которая будет служить оберткой для вызова **PARDISO**. Назовем ее **computeDecision**. Эта функция принимает на вход размеры сетки, указатель на массив, в который будет записано решение системы, и количество потоков, которое по умолчанию равно единице, а возвращает время работы решателя в секундах:

```
#include "mkl_pardiso.h"
#include "Utilities.h"

double computeDecision(int n, int m, double *u,
                       int numThreads = 1);
```

Рассмотрим реализацию этой функции, которая будет находиться в файле **PardisoCall.cpp**:

```
#include "PardisoCall.h"

double computeDecision(int n, int m, double *u,
                       int numThreads)
{
  crsMatrix A; // матрица в левой части системы
  double *b; // вектор правых частей

  // инициализация матрицы и вектора правых частей
  InitializeHCEMatrixCRS(n, m, A);
  InitializeRHVector(n, m, b);

  // приведение к массивов к фортрановскому типу
  for(int i = 0; i < A.NZ; i++)
    A.Col[i]++;
  for(int i = 0; i <= A.N; i++)
    A.RowIndex[i]++;

  // вспомогательная структура данных для работы PARDISO
  void *pt[64];
  pt[0] = 0;

  int maxfct = 1, mnum = 1,
    mtype = 2, // матрица симметрична и положительно
```

```
                // определена
    phase = 13, // факторизация и решение системы
    size = (n - 1) * (m - 1), // порядок матрицы
    *perm = new int[size], // перестановка
    nrhs = 1, // число правых частей
    msglvl = 1, // вывод статистической информации
    error = 0; // код ошибки

  int *iparm = new int[64];
  memset(iparm, 0, sizeof(int) * 64);

  clock_t start = clock();
  pardiso(pt, &maxfct, &mnum, &mtype, &phase, &size,
          A.Value, A.RowIndex, A.Col, perm,
          &nrhs, iparm, &msglvl, b, u, &error);
  double time = (double)(clock() - start) / CLOCKS_PER_SEC;

  // Печать кода ошибки работы PARDISO и количество
  // ненулевых элементов в факторе
  printf("%d;%d;", error, iparm[17]);

  delete[] perm;
  delete[] iparm;
  delete[] b;

  FreeMatrix(A);
  return time;
}
```

После того, как размеры сетки стали известны, можно инициализировать значения в матрице и векторе правой части с помощью уже реализованных функций. Отметим, что выделение памяти происходит внутри вспомогательных функций, а освобождение – внутри функции, их использующей. Как было описано выше, PARDISO работает с массивами в стиле языка FORTRAN, поэтому необходимо, чтобы соблюдалась индексация с единицы в массивах **`Col`** и **`RowIndex`**. После завершения подготовительных работ нужно включить вызов решателя, например, в приведенном в главе **Ошибка! Источник ссылки не найден.** виде. После вызова PARDISO станет известен код ошибки его работы и заполнение фактора, которые мы выведем на экран.

6.3. Последовательная версия №2. Реализация функции решения СЛАУ с использованием прямого решателя из MKL LAPACK

В этой главе рассмотрим другой способ решения СЛАУ – с помощью ленточного решателя. Для этого необходимо подготовить исходную матрицу в ленточном формате, описанном в разделе **Ошибка! Источник ссылки не найден.**. Формирование матрицы будет происходить в функции

`InitializeHCEMatrixBand`, располагающейся в файле **Utilities.h**. Если матрицу в левой части системы, полученной в главе 5, записать в ленточном формате, то окажется, что количество элементов в симметричной записи составляет $n(n-1)(m-1)$, так как лента содержит n диагоналей по $(n-1)(m-1)$ элементу. Из них 3 диагонали являются ненулевыми. Диагональные элементы будут иметь индексы $i \times n + n - 2, i = \overline{0,(m-1)(n-1)-1}$, элементы со значением $-\frac{1}{h^2}$ - индексы $i \times n + n - 2, i = \overline{0,(m-1)(n-1)-1}, i \neq j \times (n-1), j = \overline{0,m-2}$, и элементы со значениями $-\frac{1}{k^2}$ – индексы $i \times n, i = \overline{n-2,(m-1)(n-1)-1}$. Код функции, заполняющей матрицу, приведен ниже:

```
void InitializeHCEMatrixBand(int n, int m, double* & A)
{
  int size = (n - 1) * (m - 1);

  A = new double[size * n];
  memset(A, 0, sizeof(double) * size * n);

  double h = L1 / n, k = L2 / m;
  double inv_h2 = 1 / h / h;
  double inv_k2 = 1 / k / k;
  double diag = 2 * (inv_h2 + inv_k2);

  for(int i = 0; i < size; i++)
  {
    if(i >= n - 1)
      A[i * n] = - inv_k2;
    if(i % (n - 1) != 0)
      A[i * n + n - 2] = - inv h2;
    A[i * n + n - 1] = diag;
  }
}
```

Прототип и реализацию функции-обертки вызова ленточного решателя поместим в файлах **DpbtrsCall.h** и **DpbtrsCall.cpp**. Назовем ее **`computeDecision_Band`**. Для использования функций MKL, работающих с ленточными матрицами, в **DpbtrsCall.h** необходимо подключить заголовочный файл **mkl_lapack.h**. Также подключим **Utilities.h** для вызова вспомогательных функций.

```
#include "mkl_lapack.h"
#include "Utilities.h"

double computeDecision_Band(int n, int m, double * u);
```

Внутри функции **`computeDecision_Band`** предварительно вызовем функции, формирующие матрицу в ленточном формате, и функцию

составления вектора правой части, которая была описана в предыдущей главе.

```
double computeDecision_Band(int n, int m, double * u)
{
  double * A;
  InitializeHCEMatrixBand(n, m, A);
  double * b;
  InitializeRHVector(n, m, b);

  char uplo = 'U';
  int size = (n - 1) * (m - 1);
  int kd = n - 1;
  int ldab = n;
  int info;
  clock_t start = clock();
  dpbtrf(&uplo, &size, &kd, A, &ldab, &info);

  int nrhs = 1;
  dpbtrs(&uplo, &size, &kd, &nrhs, A, &ldab, b,
         &size, &info);
  double time = (double)(clock() - start) / CLOCKS_PER_SEC;

  for(int i = 0; i < size; i++)
    u[i] = b[i];

  delete[] b;
  delete[] A;

  return time;
}
```

Перед вызовом решателя **dpbtrs** предварительно необходимо факторизовать матрицу с помощью функции **dpbtrf**. Кратко остановимся на некоторых параметрах этих функций. Поскольку мы храним только верхний треугольник матрицы, параметру **uplo** необходимо присвоить значение **'U'**. Параметр **kd** означает количество диагоналей в матрице, расположенных над главной диагональю, и принимает значение $n-1$, параметр **ldab** хранит первую размерность массива, в который записана матрица в ленточном формате, то есть n. Параметр **nrhs** означает количество правых частей системы и принимает значение 1. Функция **dpbtrs** записывает решение системы в массив **b**, в котором изначально находился вектор правой части. Чтобы решение находилось в массиве **u**, нужно перезаписать в него полученные значения.

6.4. Параллельная версия. Реализация функции решения СЛАУ с использованием параллельного MKL PARDISO

Чтобы организовать параллельную работу MKL PARDISO, необходимо проделать следующие шаги:

- Собрать приложение с библиотеками, допускающими многопоточное выполнение, а именно **mkl_solver.lib** и **mkl_intel_thread.lib**. Для этого во вкладке **Linker→Input→Additional Dependencies** в свойствах проекта **Configuration Properties** необходимо вписать библиотеки **mkl_core.lib**, **mkl_intel_thread.lib**, **mkl_intel_lp64.lib**, **mkl_blas95_lp64.lib**.
- В свойствах проекта **Configuration Properties** нужно включить ключ **/Qopenmp** во вкладке **C/C++→Language→OpenMP Support**.

Варьировать количество параллельно работающих потоков будем с помощью функции **`mkl_set_num_threads`**. Для ее использования нужно подключить заголовочный файл **mkl.h**, в котором находится прототип. Вызов самой функции поставим внутри **`computeDecision`** и на вход передадим переменную **`numThreads`**. Значение числа потоков будем передавать через параметры командной строки. Если число потоков не задавать явно, то PARDISO самостоятельно определит оптимальное количество независимых потоков.

Таким образом, изменения произойдут внутри функций **`main`** и **`computeDecision`**. Они выделены полужирным.

```
int main(int argc, char* argv[])
{
  if(argc != 4)
  {
    printf("Invalid input parameters.\n 01_hce_PARDISO n m
p \n \t(n,m) - grid sizes, p - number of threads.\n");
    return 1;
  }

  int n = atoi(argv[1]);
  int m = atoi(argv[2]);
  int numThreads = atoi(argv[3]);
  ...
}
```

```
double computeDecision(int n, int m, double *u,
                       int numThreads)
{
  mkl_set_num_threads(numThreads);
  ...
```

```
}
```

Помимо этого, с помощью настройки параметров MKL PARDISO, приведенной в параграфе 2.3, можно использовать параллельную версию метода вложенных сечений для переупорядочивания матрицы.

7. Сравнение эффективности решения задачи при использовании MKL PARDISO и MKL LAPACK

Сравнение производительности представленных решателей будем производить с помощью программной реализации, описанной в параграфах 6.1-6.3. В качестве тестовых примеров возьмем системы, образованные сеткой, чьи размеры представимы в виде $100i \times 100i, i = \overline{1,10}$. В этом случае наименьший порядок системы равен 9801, а наибольший – 998001. Сначала сравним времена, затрачиваемые решателями PARDISO с разными способами переупорядочивания и LAPACK.

Таблица 2. Сравнение времен работы PARDISO и dpbtrs.

n	m	Порядок системы $(n-1) \times (m-1)$	PARDISO MD (сек)	PARDISO ND (сек)	dpbtrf + dpbtrs (сек)
100	100	9801	0,127	0,202	0,032
200	200	39601	0,202	0,421	0,338
300	300	89401	0,361	0,889	1,428
400	400	159201	0,674	1,731	4,576
500	500	249001	1,01	2,808	9,623
600	600	358801	1,682	3,837	21,221
700	700	488601	2,256	5,865	35,034
800	800	638401	3,24	7,924	58,317
900	900	808201	4,132	9,296	91,397
1000	1000	998001	5,365	13,446	136,083

Конфигурацию работы MKL PARDISO с методом минимальной степени и методом вложенных сечений соответственно назовем PARDISO MD (от англ. названия $minimum\ degree$) и PARDISO ND (от англ. названия $nested\ dissection$). Далее будем придерживаться подобных обозначений. Последние три столбца таблицы 2 содержат времена решения систем каждого тестируемого решателя.

Из таблицы видно, что при небольших размерах системы (до 10000 строк) выгоднее использовать ленточный решатель, в то время как при больших размерах наблюдается существенный выигрыш от использования PARDISO. Также важно отметить, что конфигурация PARDISO MD работает более чем в два раза быстрее по сравнению с PARDISO ND. Далее мы увидим, что большую часть времени PARDISO ND тратит на переупорядочивание матрицы, из чего можно сделать вывод, что реализованный в MKL PARDISO метод вложенных сечений на данном типе матриц работает медленнее метода минимальной степени.

Графики изменения времен работы с ростом порядка системы показаны на Рис. 5.

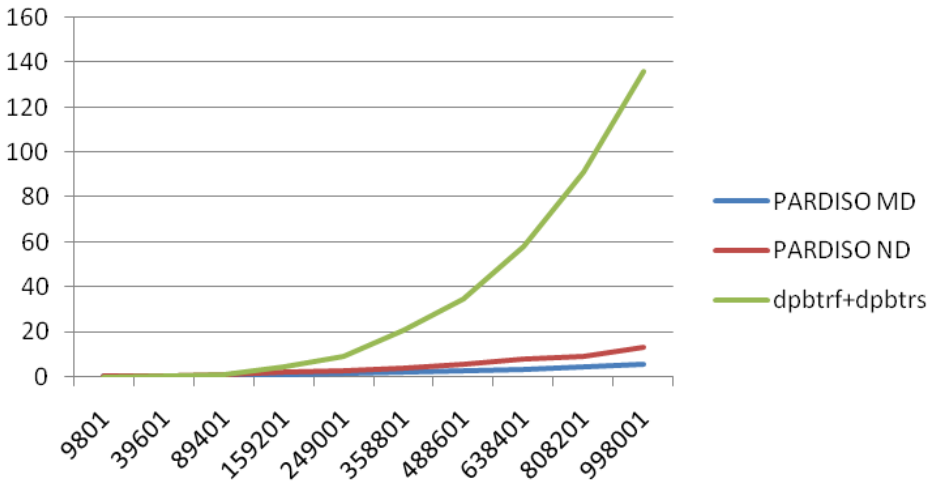


Рис. 5. Сравнение времен работы PARDISO и dpbtrs

На основе полученных времен можно подсчитать ускорение MKL PARDISO относительно тестового решателя для каждой системы.

Таблица 3. Ускорение PARDISO относительно dpbtrs

Порядок системы	PARDISO MD	PARDISO ND
9801	0,252	0,158
39601	1,673	0,803
89401	3,956	1,606
159201	6,789	2,644
249001	9,528	3,427
358801	12,617	5,531
488601	15,529	5,973

638401	17,999	7,360
808201	22,119	9,832
998001	25,365	10,121

Для системы 9801×9801 PARDISO отставал примерно в 4-6 раз от ленточного решателя. При порядке 39601 все решатели показывали почти одинаковые времена, ускорение близко к 1. Дальнейшее увеличение порядка системы показывает значительный рост ускорения. Наглядное отображение таблицы 3 представлено на Рис. 6.

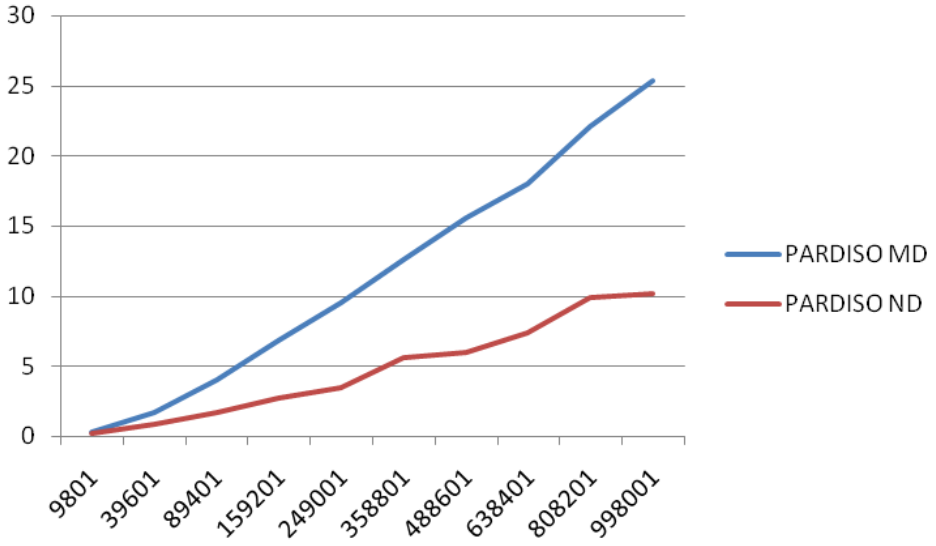


Рис. 6. Ускорение PARDISO относительно dpbtrs

Другим важным критерием эффективности работы прямого решателя СЛАУ является заполнение, возникающее при факторизации матрицы в ее левой части, поэтому сравним показатели конфигураций PARDISO MD и PARDISO ND. Заполнение фактора приведено в таблице 4.

Таблица 4. Заполнение фактора при использовании PARDISO с методами переупорядочивания MD и ND

Порядок системы	PARDISO MD	PARDISO ND
9801	0,004232274	0,004267977
39601	0,001356362	0,001349798
89401	0,000667365	0,000649365
159201	0,000409467	0,000387186
249001	0,000277474	0,000282571

358801	0,000201976	0,000200238
488601	0,000155124	0,000154148
638401	0,000122742	0,000118806
808201	9,98875E-05	9,84729E-05
998001	8,21622E-05	8,20621E-05

Приведенные данные показывают, что использование методов минимальной степени и вложенных сечений дают почти одинаковое заполнение.

Теперь рассмотрим детальнее работу MKL PARDISO на примере конфигурации с методом вложенных сечений. С помощью настройки соответствующих параметров можно получить информацию о времени работы каждой из фаз. Для краткости обозначений будем использовать те же сокращения, что и в статистической информации PARDISO:

- Fulladj – анализ снижения заполнения,
- Reorder – переупорядочивание,
- Symbfct – символическая факторизация,
- Numfct – численная факторизация,
- Solve – решение треугольных систем.

Выбор способа переупорядочивания обусловлен, во-первых, попыткой объяснить отставание конфигурации PARDISO ND, и, во-вторых, возможностью использовать параллельную версию метода вложенных сечений, что позволит проанализировать ускорение на основе полученных данных. Результаты приведены в таблице 5 и на Рис. 7.

Таблица 5. Времена работы каждой фазы решения системы с помощью конфигурации PARDISO ND

Порядок системы	Fulladj (сек)	Reorder (сек)	Symbfct (сек)	Numfct (сек)	Solve (сек)
9801	0,00016	0,04534	0,00344	0,01285	0,00092
39601	0,00067	0,21952	0,01465	0,06445	0,00476
89401	0,00155	0,51858	0,03485	0,17916	0,01202
159201	0,00388	1,02625	0,07215	0,42574	0,02827
249001	0,0061	1,73395	0,11208	0,70667	0,03927
358801	0,00703	2,28133	0,16099	1,06927	0,05721
488601	0,0101	3,547	0,22337	1,66731	0,09529
638401	0,01519	4,55604	0,33132	2,46165	0,13245
808201	0,01862	5,27582	0,37475	3,02043	0,13945

998001	0,023	7,75747	0,52908	4,33516	0,17769

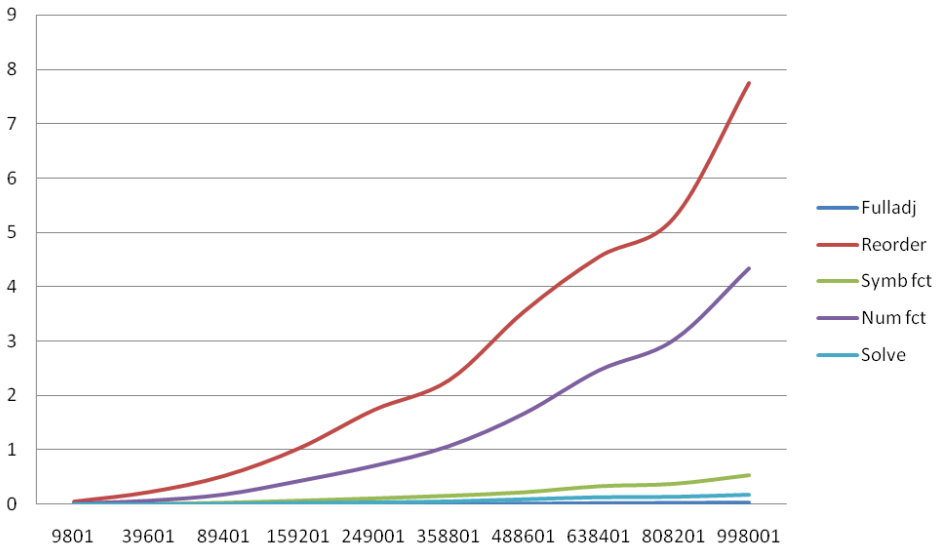


Рис. 7. Времена работы каждой фазы решения системы с помощью конфигурации PARDISO ND

Следует отметить, что суммарное время фаз, приведенных в таблице, меньше времен, приведенных в таблице 2. Данный факт можно объяснить тем, что на заполнение структур и выделение памяти тратится дополнительное время, которое не приводится в таблице с временны*ми* результатами. Полученные времена свидетельствуют о том, что наибольшую часть времени решения системы занимают переупорядочивание и численная факторизация матрицы – около 90% общего времени. Ответ на вопрос о том, насколько эффективно распараллеливается работа наиболее трудоемких этапов, рассматривается в следующей главе.

8. Анализ масштабируемости приложения при использовании MKL PARDISO для решения СЛАУ

Цель дальнейших экспериментов – исследовать эффективность MKL PARDISO при параллельном запуске. Оценивать масштабируемость будем на основании сравнения последовательной и -поточных реализаций приложения, где $p = \overline{1,8}$. При параллельном решении будем использовать OpenMP версию метода вложенных сечений.

Далее будем придерживаться следующих обозначений:

- T_s – время работы (сек) последовательной версии,
- p – количество потоков,
- T_p – время работы (сек) -поточной версии.

Для начала сравним общие времена, требуемые для решения системы. Соответствующие значения приведены в таблице 6.

Таблица 6. Времена работы последовательной и параллельных реализаций PARDISO ND

Порядок системы	T_p (сек) / T_s(сек)	p							
		1	2	3	4	5	6	7	8
9801	0,202	0,202	0,218	0,187	0,172	0,172	0,187	0,187	0,172
39601	0,421	0,452	0,374	0,296	0,296	0,265	0,296	0,281	0,265
89401	0,889	0,952	0,655	0,499	0,467	0,467	0,452	0,467	0,452
159201	1,731	1,747	1,154	0,827	0,826	0,733	0,733	0,748	0,717
249001	2,808	3,057	1,903	1,342	1,31	1,123	1,107	1,108	1,107
358801	3,837	4,212	2,543	1,81	1,794	1,544	1,529	1,529	1,482
488601	5,865	5,912	3,744	2,667	2,574	2,246	2,137	2,184	2,09
638401	7,924	7,69	4,789	3,291	3,26	2,87	2,699	2,792	2,729
808201	9,296	9,39	6,114	4,227	4,009	3,478	3,447	3,338	3,369
998001	13,446	13,882	7,877	5,538	5,428	4,554	4,43	4,523	4,336

На основе полученных данных можно отследить, насколько быстрее работает приложение с увеличением числа потоков. Для этого подсчитаем ускорение как $\frac{T_s}{T_p}$. Значения ускорения и графики его изменения относительно количества потоков для каждой размерности тестовой системы приведены в таблице 7 и на Рис. 8.

Таблица 7. Ускорение общего времени работы PARDISO ND относительно последовательной версии

$\frac{T_s}{T_p}$ / Порядок системы	p							
	1	2	3	4	5	6	7	8
9801	1,000	0,927	1,080	1,174	1,174	1,080	1,080	1,174
39601	0,931	1,126	1,422	1,422	1,589	1,422	1,498	1,589
89401	0,934	1,357	1,782	1,904	1,904	1,967	1,904	1,967

159201	0,991	1,500	2,093	2,096	2,362	2,362	2,314	2,414
249001	0,919	1,476	2,092	2,144	2,500	2,537	2,534	2,537
358801	0,911	1,509	2,120	2,139	2,485	2,509	2,509	2,589
488601	0,992	1,567	2,199	2,279	2,611	2,745	2,685	2,806
638401	1,030	1,655	2,408	2,431	2,761	2,936	2,838	2,904
808201	0,990	1,520	2,199	2,319	2,673	2,697	2,785	2,759
998001	0,969	1,707	2,428	2,477	2,953	3,035	2,973	3,101

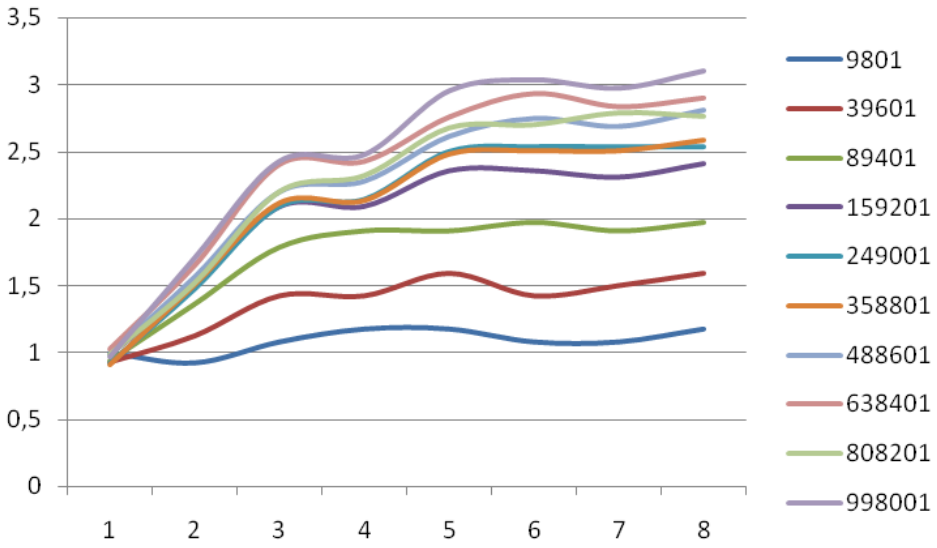


Рис. 8. Ускорение общего времени работы PARDISO ND относительно последовательной версии

Заметим, что наибольший рост ускорения наблюдается при изменении числа потоков от 3 до 5. Дальнейшее увеличение этого числа дает небольшой прирост производительности. Работа MKL PARDISO складывается из нескольких фаз, поэтому, говоря о масштабируемости, необходимо знать, как работает и распараллеливается каждая из них. Зафиксируем порядок системы на значении 998001 и проведем расчеты для всех фаз решателя. В таблице 8 приведены времена работ каждой фазы.

Таблица 8. Времена работ фаз PARDISO ND в последовательной и параллельных реализациях

Фаза	T_p (сек) / T_s (сек)	p							
		1	2	3	4	5	6	7	8

Fulladj	0,023	0,02	0,02	0,022	0,021	0,022	0,022	0,023	0,021
Reorder	7,757	7,637	4,076	2,553	2,515	1,823	1,799	1,828	1,831
Symbfct	0,529	0,785	0,504	0,447	0,425	0,494	0,431	0,460	0,445
Numfct	4,335	4,409	2,361	1,822	1,689	1,539	1,431	1,387	1,365
Solve	0,178	0,177	0,125	0,107	0,074	0,079	0,076	0,080	0,079

Аналогично посчитаем ускорение относительно последовательной версии для каждой фазы.

Таблица 9. Ускорение фаз PARDISO ND относительно последовательной версии

$\frac{T_S}{T_p}$ Фаза	p							
	1	2	3	4	5	6	7	8
Fulladj	1,137	1,162	1,051	1,091	1,063	1,064	0,987	1,083
Reorder	1,016	1,903	3,039	3,085	4,256	4,313	4,243	4,237
Symbfct	0,674	1,050	1,184	1,246	1,070	1,228	1,151	1,188
Numfct	0,983	1,837	2,379	2,567	2,816	3,029	3,125	3,177
Solve	1,003	1,426	1,663	2,386	2,249	2,344	2,208	2,261

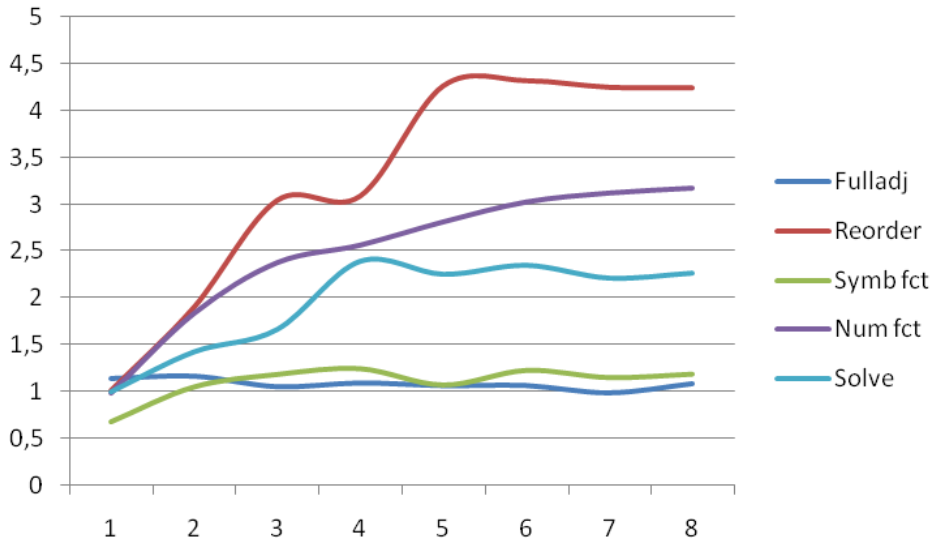


Рис. 9. Ускорение фаз PARDISO ND относительно последовательной версии

Из Рис. 9 видно, что графики ускорения параллельного метода вложенных сечений и решения треугольных систем по характеру роста напоминают графики, представленные на Рис. 9, где также наблюдались сильный рост при переходе от 3 до 5 потоков и дальнейшее горизонтальное движение. Более хорошая масштабируемость наблюдается для фазы численной факторизации, график ускорения которой равномерно возрастает с ростом числа потоков. Графики анализа разреженности и символической факторизации слабо колеблются около 1, из чего можно сделать вывод о том, что эти фазы либо не распараллеливаются, либо алгоритмы, их реализующие, обладают небольшим запасом внутреннего параллелизма.

Суммируя полученные результаты, можно сказать, что использование параллельного MKL PARDISO оправдано, и уже на 5 потоках можно достичь троекратного ускорения.

9. Контрольные вопросы

1. Объясните метод решения СЛАУ, который реализован в пакете MKL PARDISO?
2. Когда оправдано использование решателя MKL PARDISO?
3. Покажите, что переупорядочивание не изменяет знакоопределенность матрицы.

10. Дополнительные задания

1. Определите порядок аппроксимации разностной схемы (8).
2. Оцените трудоемкость метода Холецкого для случая пятидиагональной матрицы, рассматриваемой в лабораторной работе.
3. Разработайте метод Гаусса для решения СЛАУ с пятидиагональной матрицей, которая рассмотрена в работе. Реализуйте последовательную и параллельную реализацию разработанного метода. Оцените эффективность применения разработанной последовательной реализации к решению стационарной задачи распределения температур в прямоугольной пластине. Проведите анализ масштабируемости приложения при использовании параллельной реализации метода Гаусса.
4. Оценить трудоемкость фаз решения СЛАУ (переупорядочивание, символическая и численная факторизация и т.д.).
5. Вычислите аналитически заполнение фактора для матрицы теплопроводности без переупорядочивания и сравните его со

значением заполнения, которое вычислит PARDISO при факторизации матрицы с прямой перестановкой.

11. Литература

11.1. Использованные источники информации

1. Тихонов А.Н., Самарский А.А. Уравнения математической физики. – М.: Наука, 1977.
2. Самарский А.А., Гулин А.В. Численные методы. – М.: Наука, 1989.
3. Вержбицкий В.М. Численные методы. – М.: Высшая школа, 2001.
4. Байков В.А., Жибер А.В. Уравнения математической физики. – Москва-Ижевск: Институт компьютерных исследований, 2003.
5. Писсанецки С. Технология разреженных матриц. – М.: Мир, 1988.
6. Дж. Голуб, Ч. Ван Лоун. Матричные вычисления. – М.: Мир. 1999.

11.2. Дополнительная литература

7. James R. Bunch and Linda Kaufman, Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems. 1977. Mathematics of Computation, volume 31, number 137, pp. 163-179.

11.3. Ресурсы сети Интернет

8. Список математических пакетов и библиотек [http://en.wikipedia.org/wiki/List_of_numerical_libraries].
9. Официальная страница библиотеки Intel® Math Kernel Library [http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/].
10. Страница учебного курса «Параллельные численные методы» [http://www.hpcc.unn.ru/?doc=491].