УДК 519.6:519.85

Е. Ю. Карпенко

Институт проблем моделирования в энергетике им. Г. Е. Пухова НАН Украины, г. Киев

ПАРАЛЛЕЛЬНОЕ РЕШЕНИЕ ИНТЕГРАЛЬНЫХ УРАВНЕНИЙ ФРЕДГОЛЬМА I РОДА МЕТОДОМ РЕГУЛЯРИЗАЦИИ ТИХОНОВА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ MPI

В статье рассмотрен программный подход к решению интегрального уравнения Фредгольма I рода методом регуляризации Тихонова, который заключается в использовании многопроцессорных компьютерных систем и стандарта параллельного программирования MPI.

Ключевые слова: *метод регуляризации Тихонова, интегральное уравнение Фредгольма, MPI, ScaLAPACK, mpich.*

Введение и постановка задачи. Целый ряд прикладных задач, таких как, восстановление непрерывного спектра в обратной задаче спектроскопии, редукция протяженных сигналов, восстановление сигналов в системе не являющейся динамической, распад клеток и радиоактивных элементов, вентиляция в легких и другие описываются интегральным уравнением Фредгольма I рода [1]:

$$\int_a^b K(x,s)y(s)ds = f(x), \quad c \le x \le d\ , \tag{1}$$

где $K(x,s)$, с математической точки зрения является ядром интегрального уравнения (с физической, или технической – аппаратная функция, характеристика направленности и т.д.), $f(x)$ – правая часть (выходной сигнал, сканирующая функция, индикаторный процесс и т.д.), $y(s)$ – искомая функция (сигнал на входе и т.д.).

Уравнение (1) занимает особое положение среди других интегральных уравнений, по причине своей некорректности (нарушается устойчивость решения по Адамару). Например, если решать уравнение (1) методом квадратур, заменяя интеграл конечной суммой по формуле трапеций, и решая полученную СЛАУ, то вместо истинного решения, как правило, получается знакопеременная "пила" большой амплитуды, которая при подстановке в интеграл, тем не менее, дает правую часть уравнения. При этом, чем меньше шаг разбиения и чем, казалось бы, точнее аппроксимируется интегральное уравнение посредством СЛАУ, тем грубее решение (больше амплитуда "пилы"). Аналогичная неустойчивость имеет место при решении уравнения (1) методами собственных функций, итераций, проекционными методами [2].

В настоящее время известно большое количество эффективных методов решения интегральных уравнений Фредгольма I-го рода. В основе большинства таких методов лежит сведение интегрального уравнения (каким либо способом) к СЛАУ. При этом если требуется повысить точность решения исходного интегрального уравнения, увеличив размерность СЛАУ, то это, в свою очередь, приведет к понижению скорости решения системы, а в некоторых случаях и к дефициту компьютерного ресурса (памяти). В таких случаях целесообразно использовать современные эффективные средства распараллеливания.

Метод регуляризации Тихонова. Метод регуляризации Тихонова является продолжением развития метода наименьших квадратов (МНК) Гаусса и метода псевдо-обратной матрицы (МПОМ) Мура-Пенроуза (дающего нормальное решение).

Рассмотрим сначала сущность метода применительно к операторному уравнению:

$$Ay = f, \quad y, f \in L_2, \tag{2}$$

где A – линейный оператор, f – заданная правая часть, а y – искомое решение.

В методе регуляризации Тихонова ставятся два условия: условие минимизации невязки (как в МНК) и условие минимизации нормы решения (как в МПОМ). Получаем задачу условной минимизации, которая решается методом неопределенных множителей Лагранжа:

$$\|Ay - f\|_{L_2}^2 + \alpha \|y\|_{L_2}^2 = \min_y, \tag{3}$$

где $\alpha > 0$ параметр регуляризации, играющий роль неопределенного множителя Лагранжа. Отсюда вытекает уравнение Тихонова:

$$(\alpha E + A^* A) y_\alpha = A^* f, \tag{4}$$

вследствие чего мы получаем операторное уравнение второго рода.

Метод регуляризации Тихонова устойчив, т.е. выполняется 3-й пункт корректности по Адамару и эта устойчивость заключается в следующем. Оператор $A^* A$ является положительно определенным, поэтому все его собственные значения вещественны и неотрицательны. Наличие же слагаемого αE увеличивает спектр на α, вследствие чего оператор $\alpha E + A^* A$ становится обратимым и решение представимо в виде:

$$y_\alpha = (\alpha E + A^* A)^{-1} A^* f. \tag{5}$$

Применительно к интегральному уравнению Фредгольма I рода

$$Ay \equiv \int_a^b K(x, s) y(s) ds = f(x), \quad c \le x \le d \tag{6}$$

соотношение (4) приобретает вид интегрального уравнения Фредгольма II рода с положительно определенным ядром:

$$\alpha y_\alpha(t) + \int_a^b R(t,s) y_\alpha(s) ds = F(t), \quad a \le t \le b, \tag{7}$$

где

$$R(t,s) = R(s,t) = \int_c^d K(x,t) K(x,s) dx, \tag{8}$$

$$F(t) = \int_c^d K(x,t) f(x) dx. \tag{9}$$

Численный алгоритм. Достаточно эффективным при численном решении интегрального уравнения (7) является квадратурный метод.

Пусть правая часть уравнения $f(x)$ задана на неравномерной сетке узлов $\{x_i\}_{i=1}^l$:

$$c = x_1 < x_2 < \ldots < x_l = d,$$

а решение $y_\alpha(s)$ ищется на неравномерной сетке $\{s_j\}_{j=1}^n$, совпадающей с t – сеткой:

$$a = s_1 = t_1 < s_2 = t_2 < \ldots < s_n = t_n = b.$$

Распишем интеграл в (7) по квадратурной формуле трапеций:

$$\alpha y_k + \sum_{j=1}^n r_j R_{kj} y_j = F_k, \qquad k = \overline{1,n}, \tag{10}$$

где $y_k = y_\alpha(t_k)$, $y_j = y_\alpha(s_j)$, $R_{kj} = R(t_k, s_j)$, $F_k = F(t_k)$. Аналогичным образом интегралы в (8) и (9) аппроксимируем конечными суммами по квадратурной формуле:

$$R_{kj} = R_{jk} = \sum_{i=1}^l p_i K_{ik} K_{ij}, \qquad k, j = \overline{1,n}, \tag{11}$$

$$F_k = \sum_{i=1}^l p_i K_{ik} f_i, \qquad k = \overline{1,n}, \tag{12}$$

где $K_{ik} = K(x_i, t_k)$, $K_{ij} = K(x_i, s_j)$, $f_i = f(x_i)$, а r_j и p_i – коэффициенты квадратурных формул.

Запись (10) является СЛАУ относительно y_j, $j = \overline{1,n}$. При больших n время, затраченное на решение системы, играет заметную роль.

Стандарт MPI. Наиболее распространенной технологией программирования для параллельных компьютеров с распределенной памятью в настоящее время является MPI [3]. Основным способом взаимодействия параллельных процессов в таких системах является передача сообщений друг другу. Это и отражено в названии данной технологии Massage Passing Interface (интерфейс передачи сообщений).

В модели программирования, которую поддерживает MPI, программа порождает несколько процессов, взаимодействующих между собой с помощью обращений к подпрограммам передачи и приема сообщений. Все процессы порождаются один раз, образуя параллельную часть программы. При этом каждый процесс работает в своем адресном пространстве, никаких общих переменных или данных в MPI нет. Обычно, при инициализации MPI-программы создается фиксированный набор процессов, причем каждый процесс выполняется на своем процессоре. В этих процессах могут выполняться разные программы, поэтому модель программирования MPI иногда называют MPMD-моделью (Multiple Program Multiple Data – множество программ множество данных), в отличие от SPMD-модели, где на каждом процессоре выполняются только одинаковые задачи.

Для локализации взаимодействия параллельных процессов программы можно создавать группы процессов, предоставляя им отдельную среду для процессов – коммуникатор. Состав образуемых групп произволен. Группы могут полностью совпадать, входить одна в другую, не пересекаться или пересекаться частично. Процессы могут взаимодействовать только внутри некоторого коммуникатора, сообщения, отправленные в разных коммуникаторах, не пересекаются и не мешают друг другу. При старте программы всегда считается, что все порожденные процессы работают в рамках всеобъемлющего коммуникатора, имеющего предопределенное имя MPI_COMM_WORLD.

Каждый процесс MPI программы имеет в каждой группе, в которую он входит, уникальный атрибут – номер процесса, который является целым неотрицательным числом. Как уже было отмечено, основным способом общения процессов между собой является явная посылка сообщений. Каждое сообщение имеет несколько атрибутов, в частности, номер процесса-отправителя, номер процесса-получателя, идентификатор сообщения и другие. По идентификатору процесс, принимающий сообщение, например, может различить два сообщения, пришедшие к нему от одного и того же отправителя.

MPICH (MPI CHameleon) представляет собой одну из реализаций спецификации MPI, которая поддерживает работу на большом числе платформ и с различными коммуникационными интерфейсами, в том числе TCP/IP.

В настоящее время для решения стандартных задач, таких как, например, задачи линейной алгебры, разработаны библиотеки, построенные на основе стандарта MPI.

Использование библиотеки ScaLAPACK. ScaLAPACK является библиотекой высокоэффективных подпрограмм линейной алгебры разработанных для компьютерных систем с распределенной памятью, на основе стандартов параллельного программирования PVM или MPI [4].

Иерархия программных компонент библиотеки ScaLAPACK представлена на *рис. 1*.

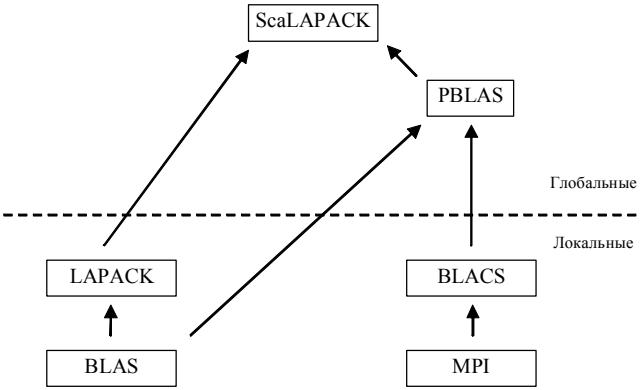

Рис. 1. *Иерархия программных компонент библиотеки ScaLAPACK*

LAPACK (Linear Algebra PACKage) – библиотека, содержащая подпрограммы для решения систем линейных алгебраических уравнений, метода наименьших квадратов, решения задач на собственные значения и вычисления сингулярных разложений.

BLAS (Basic Linear Algebra Subprograms) – содержит подпрограммы для наиболее распространенных операций линейной алгебры: скалярное произведение, матрично-векторное умножение, умножение матрицы на матрицу.

PBLAS (Parallel Basic Linear Algebra Subprograms) – для упрощения библиотеки ScaLAPACK и в связи с тем, что полезные инструменты не вошли в библиотеку LAPACK была разработана параллельная часть библиотеки BLAS.

BLACS (Basic Linear Algebra Communication Subprograms) – библиотека передачи сообщений, приспособленная для линейной алгебры. Вычислительная модель разработана для одномерной и двумерной решетки процессов, в которой каждый процесс содержит часть матрицы или вектора.

Процедуры и функции из локальных компонентов вызываются на одном процессе и их аргументы также хранятся только на одном процессе. В свою очередь глобальные компоненты содержат синхронизированные параллельные подпрограммы, чьи аргументы представляют собой матрицы и векторы, распределенные по процессам.

В основу подпрограмм ScaLAPACK положены блочные алгоритмы, которые разработаны для минимизации передачи данных между различными уровнями памяти.

Вызов любой подпрограммы из библиотеки ScaLAPACK состоит из 4-х базовых шагов:

1) инициализация решетки процессов;
2) распределение матрицы по решетки процессов;
3) вызов процедуры ScaLAPACK;
4) освобождение решетки процессов.

Программный алгоритм. Для программной реализации в среде Microsoft Visual Studio 2005 Express Edition с использованием языка Visual Fortran 10 (Trial Version) был написан программный модуль, рассчитанный на решение задачи в компьютерной системе, состоящей из произвольного числа процессоров. Для организации стандарта параллельного программирования MPI использовался комплекс MPICH2.

Алгоритм программной реализации решения интегрального уравнения Фредгольма I рода методом регуляризации Тихонова может быть представлен в виде последовательности следующих действий:

1) Описание глобальных и локальных переменных, массивов, подпрограмм, функций. Задание размеров матрицы (N), блока разбиения (NB), максимального размера подматрицы ($MAXA$).

2) Инициализация BLACS:
call blacs_pinfo(iam, nprocs)

3) Вычисление сетки процессов наиболее приближенной к квадратной:
nprow = int(sqrt(real(nprocs)))
npcol = nprocs/nprow

4) Инициализация решетки процессов:
call blacs_get(-1, 0, ictxt)
call blacs_gridinit(ictxt, 'row-major', nprow, npcol)
call blacs_gridinfo(ictxt, nprow, npcol, myrow, mycol)
Процедура BLACS_GRIDINIT инициализирует решетку процессов с идентификатором ICTXT размера NPROW x NPCOL. После инициализации решетки процессов функция BLACS_GRIDINFO позволит узнать координаты процесса, вызвавшего ее.

5) Если процесс не попал в сетку он “отдыхает”
if(myrow.ge.nprow.or.mycol.ge.npcol) go to 500

6) Инициализация массивов-дескрипторов для матриц A и B :
call descinit(desca, n, n, nb, nb, 0, 0, ictxt, maxa, info)
call descinit(descb, n, 1, nb, 1, 0, 0, ictxt, maxa, info)

Все глобальные матрицы перед вызовом процедуры ScaLAPACK должны быть разбросаны по решетки процессов, для этого каждой матрице назначается так называемый *массив-дескриптор*, состоящий из 9 элементов.

7) Вычисление реальных размеров матрицы на процессоре:

```
np = numroc(n, nb, myrow, 0, nprow)
nq = numroc(n, nb, mycol, 0, npcol)
```

8) Вычисление начальных условий (шаг разбиения, сетка разбиения, квадратурные коэффициенты)

9) Вычисление правой части системы (матрица B) и распределение ее по решетке процессов (участвует только 1-я колонка сетки процессов). Правая часть интегрального уравнения берется из файла F.dat, который формируется служебной программой BUILD_F.

```
if (mycol.eq.0) then
open(unit=myrow, file='\\k0\mpi\projects\solve\debug\f.dat')
        read (myrow, * ) f
        do jj = 1, np
            j=indxl2g(jj, nb, myrow, 0, nprow)
            do i = 1, n
                b(jj)=b(jj)+at(i)*ker(x(i),x(j))*f(i)
            end do
        end do
end if
```

10) Вызов подпрограммы формирования матрицы A:

```
call matinit (a, desca, n, nb, maxa, x, at, myrow, mycol, nprow, npcol, np, nq)
```

Данная подпрограмма служит для формирования матрицы A на процессах участвующих в программе. Следует отметить, что на каждом процессе формируется своя часть матрицы, которая отвечает сетке разбиения процессов и сетке разбиения самой матрицы. На процессах матрица A должна задаваться как одномерный массив (нумерация ведется по столбцам).

11) Формирования матрицы A на процессах согласно формулам (10)-(11):

```
l=0
do ii=1, nq
        i=indxl2g(ii, nb, mycol, 0, npcol)
        do jj=1, np
            j=indxl2g(jj, nb, myrow, 0, nprow)
            l=l+1
            temp=0
                do k=1,n
            temp=temp+at(k)*ker(x(k),x(i))*ker(x(k),x(j))
                end do
```

```
                                    aa(l)=at(i)*temp
                                    if (i.eq.j) then
                                            aa(l)=aa(l)+alpha
                                    end if
                    end do
            end do
```

Служебная функция **INDXL2G** позволяет перейти от локальных индексов в распределенной матрице к глобальным индексам в исходной.

12) Вызов подпрограммы из ScaLAPACK для решения СЛАУ методом LU разложения

```
call pdgesv(n, 1, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
```

13) Посылка главному процессу (“мастеру”) решения от процессов первой колонки сетки

```
if (myrow.gt.0.and.mycol.eq.0) then
            call mpi_send (b, np, mpi_double_precision, 0, 1, mpi_comm_world, ierr)
end if
```

14) Часть программы, выполняемая главным процессом (“мастером”) – сбор решения. Получение решения от других процессов первой колонки.

```
do k=1,np
            j=indxl2g(k, nb, myrow, 0, nprow)
            y(j)=b(k)
end do
do i=1,nprow-1
            source=int(npcol*i)
            call mpi_recv (temp, np, mpi_double_precision, source, 1, mpi_comm_world, status, ierr)
            row_of_send=status(mpi_source)/nprow
            do k=1,np
                        j=indxl2g(k, nb, row_of_send, 0, nprow)
                        y(j)=temp(k)
            end do
end do
```

15) Закрытие BLACS:

```
call blacs_gridexit(ictxt)
call blacs_exit(0)
```

16) Для компиляции программы и для создания исполняемого файла необходимо пролиньковать следующие библиотеки:

mpi.lib, fmpich2.lib, SCALAPACKd.lib, BLACSd.lib, BLACS_Finitd.lib, LAPACKd.lib, MATGENd.lib, extrasd.lib, BLASd.lib.

Выводы. Заметный эффект от распараллеливания начинает наблюдаться при решении систем с 1000 и более неизвестными. На кла-

стерных системах ситуация еще хуже. Разработчики пакета ScaLAPACK для многопроцессорных систем с приемлемым соотношением между производительностью узла и скоростью обмена дают следующую формулу для количества процессоров, которое рекомендуется использовать при решении задач линейной алгебры:

$$P = M \times N / 10^6, \qquad (13)$$

где M, N – размерность матрицы

Или, другими словами, количество процессоров должно быть таково, чтобы на процессор приходился блок матрицы размером примерно 1000x1000. Эта формула, конечно, носит рекомендательный характер, но, тем не менее, наглядно иллюстрирует, для задач какого масштаба разрабатывался пакет ScaLAPACK. Рост эффективности распараллеливания при увеличении размера решаемой системы уравнений объясняется очень просто: при увеличении размерности решаемой системы уравнений объем вычислительной работы растет пропорционально n^3, а объем обменов между процессорами пропорционально n^2. Это снижает относительную долю коммуникационных затрат при увеличении размерности системы уравнений.

При решении тестовых примеров были сформированы матрицы размерности (512x512, 1024x1024, 2048x2048), что позволило проверить эффективность решения системы на одном, двух и четырех процессорах. Как оказалось, рассмотренный в работе подход к расспараллеливанию формирования СЛАУ, получаемой при решении интегрального уравнения Фредгольма I рода методом регуляризации Тихонова позволяет существенно повысить скорость формирования матрицы системы (n-кратное увеличение). Однако скорость решения получаемой СЛАУ падает, что вызвано относительно небольшими размерами матрицы.

Список использованной литературы:

1. Сизиков В. С. Устойчивые методы обработки результатов измерений. Учебное пособие. – СПб.: СпецЛит, 1999. – 240 с.
2. Верлань А. Ф., Сизиков В. С. Интегральные уравнения: методы, алгоритмы, программы: Справочное пособие. – К.: Наукова думка, 1986. – 544 с.
3. Немнюгин С. А., Стесик О. Л. Параллельное программирование для многопроцессорных вычислительных систем – СПб.: БХВ-Петербург, 2002. – 400 с.
4. Интернет ресурс: http://www.netlib.org

The article reviewed the program approach to solving Fredholm integral equation I kind by Tikhonov regularization method, which is used a multiprocessor computer systems, and standard of parallel programming MPI.

Key words: *Tikhonov regularization method, Fredholm integral equation, MPI, ScaLAPACK, mpich.*

Отримано: 05.06.2008