Оптимизация решения линейных алгебраических систем МКЭ

Optimization of the solution of FEM linear algebraic systems

Ключевые слова: распараллеливание программ — program parallelization; метод конечных элементов — finite element method.

Обсуждается ускорение работы программ, достигаемое распараллеливанием. Описан доступный обычному пользователю набор инструментов программного распараллеливания. Оценена его эффективность на типовой задаче метода конечных элементов на различных платформах. Рассмотрены область применения и возможные трудности распараллеливания.

This article looks at programs acceleration by dint of parallelization. It shows a program parallel toolset, which can be used by a common user. Also article contains estimation of its effectiveness in the typical problem of finite element method on different platforms. You can also see material about scope of these methods and some problems, which can appear.

ВИННИК / VINNIK Р.
Петр Михайлович
(sigure@rambler.ru)
кандидат физико-математических наук,
доцент кафедры высшей математики
Балтийского Государственного технического
(Военмех) университета
им. Д.Ф. Устинова,
Санкт-Петербург

ВИННИК / VINNIK Т.
Татьяна Викторовна
(vinnik.tv92@gmail.com)
кандидат технических наук,
доцент кафедры высшей математики
Балтийского Государственного технического
(Военмех) университета
им. Д.Ф. Устинова,
Санкт-Петербург

ПОПОВ / POPOV А.
Александр Михайлович
(ampopov37@yandex.ru)
кандидат физико-математических наук,
доцент кафедры высшей математики
Балтийского Государственного технического
(Военмех) университета
им. Д.Ф. Устинова,
Санкт-Петербург

Одним из основных методов численного моделирования технических задач является метод конечных элементов (МКЭ) [2]. В его основе лежит идея разбиения сплошной среды на совокупность конечных элементов (КЭ) с аппроксимацией непрерывных величин дискретной моделью, строящейся на множестве конечно-элементных функций. Существенной частью МКЭ является решение системы линейных алгебраических уравнений с глобальной матрицей жесткости. Контроль за надежностью и точностью результатов МКЭ основан на отслеживании отсутствия вырождения КЭ, а математически – на отслеживании поведения спектра матрицы жесткости.

Решение полной проблемы собственных значений для произвольных матриц может быть осуществлено различными методами, среди которых такие, например, как LR-алгоритм, QR-алгоритм, метод вращений Якоби [4]. Для матриц, имеющих специальный вид – симметричный или диагональный – за счет особых методов решения проблемы собственных значений можно получить значительный выигрыш по времени.

Необходимость все более точного решения технических задач приводит к увеличению количества КЭ в МКЭ, что означает соответствующее возрастание размеров глобальной матрицы жесткости. Общетехнические расчетные инженерные пакеты способны оперировать с сотнями миллионов КЭ, т.е. с глобальными матрицами жесткости таких же порядков. Однако столь высокие показатели достигаются на суперкомпьютерах. Рядовые же потребители не имеют возможности вести расчеты на суперкомпьютерах и опираются

в своей работе на «домашние» компьютеры. При этом приемлемое время работы программы не должно превосходить нескольких суток. Анализ сложной технической задачи невозможен без расчета многих различных вариантов. Это еще больше увеличивает необходимое для анализа время. Для сокращения времени теоретически рекомендуется осуществить распараллеливание задачи, именно с помощью которого и достигаются приемлемые скорости решения упомянутых задач МКЭ на суперкомпьютерах. Из закона Амдала [1] следует, что максимальное уменьшение времени, достигаемое распараллеливанием, ограничено временем, первоначально занимаемым распараллеливаемой частью. Поэтому чем большая часть программы может быть распараллелена, тем больший выигрыш во времени может быть получен. Весомым аргументом в пользу распараллеливания задач МКЭ является то, что применение МКЭ приводит к системе линейных алгебраических уравнений, а многие алгоритмы задач линейной алгебры, в частности – решения систем линейных алгебраических уравнений и многие алгоритмы решения полной проблемы собственных значений хорошо поддаются распараллеливанию, т.е. теоретически можно рассчитывать на значительное уменьшение времени работы программы.

К сожалению, на пути практической реализации распараллеливания рядовым пользователем встают две проблемы. Во-первых, в отличие от суперкомпьютера «домашний» компьютер имеет заведомо небольшое количество процессоров (сегодня чаще всего один с несколькими ядрами), поэтому возможности применения технологий распараллеливания, существенно использующих наличие именно нескольких процессоров, заведомо серьезно ограничены. Во-вторых, рядовой пользователь, как правило, не является специалистом по распараллеливанию программ и алгоритмов, поэтому не может даже надеяться самостоятельно достичь наилучшего распараллеливания. Однако в том случае, когда программа пользователя написана на широко распространенных языках высокого уровня, например – Fortran-77 или C++, пользователь может выполнить распараллеливание с помощью команд языка MPI. Однако освоение MPI требует определенных навыков и затрат усилий и времени. Технология OpenMP также дает рядовому пользователю возможность произвести распараллеливание своей программы, написанной на языке Fortran или C++, однако в отличие от MPI, использование OMP доступно рядовому пользователю почти без подготовки, так как просто добавляя в свою программу директивы OpenMP, он производит распараллеливание. Для всего этого пользователь должен, во-первых, обладать общим представлением о том, какие операции теоретически могут быть хорошо распараллелены, во-вторых, представлять себе, какие именно блоки его программы наиболее затратны по времени исполнения, т.е. где распараллеливание наиболее целесообразно, а в-третьих – уметь «отдать команду распараллелить» тот или иной кусок программы. Так как такая «отдача команды» делается всегда одинаково (много времени для исполнения используют блоки, содержащие многократно повторяющиеся действия, например – циклы, распараллеливание которых пояснено ниже), у рядового пользователя появляется возможность распараллелить свою последовательную программу и, проведя сравнительные расчеты, оценить наличие и размер полученного выигрыша по времени. Как указывается в [3], такое распараллеливание имеет крайне ограниченные возможности, но для рядового пользователя выбор делается между отсутствием распараллеливания вообще и таким «автоматическим» распараллеливанием. Понятно, что при наличии пусть скромного выигрыша во времени пользователь заинтересован даже в таком распараллеливании. А конкретно для задач МКЭ даже такое распараллеливание по вышеизложенному может и должно давать довольно существенный выигрыш.

Для измерения выигрыша по времени целесообразно использовать следующую величину [3]: практическое ускорение = (время работы последовательного алгоритма при наихудших начальных данных) / (время работы параллельной версии этого алгоритма при тех же начальных данных). Для задач МКЭ конкретного пользователя диапазон возможных начальных данных определяется классом рассматриваемых задач и поэтому сравнительно узок. Кроме того, конкретного пользователя интересует не эффективность проделанного им распараллеливания «вообще», а реально достигаемое ускорение работы его программы на его возможных данных. Поэтому вместо рассмотрения наихудших возможных данных при вычислении практического ускорения представляется разумным использование рандомизированных начальных данных и измерение «среднего практического ускорения». Также использование среднего практического ускорения позволяет применить весь стандартный аппарат математической статистики для построения выводов о достигнутой эффективности распараллеливания. Рассмотрим превращение последовательной программы в параллельную

Размер матрицы, условия испытаний	Среднее время работы программы без распараллеливания, секунд	Среднее время работы программы с распараллеливанием, секунд	Среднее практическое ускорение, количество раз
24×24, двухядерный процессор (по 7 измерениям)	0,959	0,870	1,10
66×66, двухядерный процессор (по 11 измерениям)	32,6	23,1	1,41
66×66, четырехядерный процессор (по 4 измерениям)	20,7	16,6	1,25
210×210, двухядерный процессор (по 3 измерениям)	18229	11266	1,62

при помощи OpenMP на примере перемножения матриц (это только один из способов распараллелить этот процесс, который не претендует на оптимальность).

Приведен кусок кода программы на языке C++, распараллеленной при помощи OpenMP – цикл, вычисляющий матрицу A2 – произведение матриц A и J1. Команда «распараллелить» – это первая строка блока. В ней указывается (omp parallel for), что должен быть распараллелен цикл «for» и массивы A2, A, J1 должны быть общими для параллельных потоков, а переменные j, k, td1 – частными, т.е. локальными для каждого потока. Необходимо отдельно подчеркнуть, что для корректной работы распараллеленного отрезка программы следует подключить файл omp.h в начале программы и разрешить использование OpenMP в свойствах проекта (на вкладке Project).

```
#pragma omp parallel for shared(A2,A,J1) private(j,k,td1)
        for (int i=1;i<=J1.dim;i++)
        {
                for (j=1;j<=J1.dim;j++)
                {
                        td1=0;
                        for (k=1;k<=A.dim;k++)
                        {
                                td1+=J1.m[i][k]*A.m[k][j];
                        }
                        A2.m[i][j]=td1;
                }
        }
```

Были проведены сравнительные испытания времени работы программы, реализующей метод вращений Якоби нахождения всех собственных чисел матрицы без распараллеливания и с ним. В качестве начальных данных брались матрицы размеров 66–66 и 210–210 специального вида – глобальные матрицы жесткости из МКЭ. Осуществлялась проверка корректности работы программы сравнением полученного программой спектра матрицы и спектра, найденного в стандартном математическом пакете. Испытания проводились на компьютерах с процессором Intel Core 2 Duo T9400 с частотой 2,53 GHz и памятью 4 Gb (индекс производительности Windows 5,4) и процессором Intel Core i3-2100 с частотой 3,10 GHz и памятью 4 Gb (индекс производительности Windows 5,9). Результаты расчетов приведены в таблице.

Следует заметить, что данные таблицы, показывающие рост ускорения с ростом размера матрицы, находятся в согласии с теоретическими положениями. Действительно, матрица 24–24 для нахождения спектра методом Якоби требует всего 816 итераций, тогда как матрица 210–210 требует 63 304 итераций, а распараллеленное умножение матриц используется как раз в каждой итерации метода – таким образом, доля распараллеленного участка программы в общем времени работы программы возрастает, следовательно, должно возрастать и ускорение.

Директивы распараллеливания при определенных условиях могут создать существенный выигрыш в скорости исполнения, но ими надо пользоваться осторожно. При некорректном распараллеливании (например, когда итерации одного потока зависят от итераций другого или неправильном указании разделяемых и частных переменных) программа может перестать выдавать правильные значения. Рассмотрим это на корректном примере. В методе Якоби в некоторый момент нам необходимо искать максимум верхнетреугольной матрицы. Матрицы могут быть значительных размеров, поэтому логично распараллелить этот процесс и участок:

```
for (int i=1;i<=A.dim-1;i++)
        {
                for (j=i+1;j<=A.dim;j++)
                {
                        if ((td1<=abs(A.m[i][j]))&&(i-j!=0))
                        {
                                im=i;   jm=j;   td1=abs(A.m[i][j]);
                        }
                }
        }
```

заменить на такой:

```
#pragma omp parallel for shared(A,td1,im,jm) private(j)
        for (int i=1;i<=A.dim-1;i++)
        {
                for (j=i+1;j<=A.dim;j++)
                {
                        if ((td1<=abs(A.m[i][j]))&&(i-j!=0))
                        {
                                im=i;   jm=j;   td1=abs(A.m[i][j]);
                        }
                }
        }
```

Второй из этих кусков кода работает некорректно, так как итерации внутреннего цикла зависят от счетчика i, который по умолчанию является частной переменной потоков. Но даже при корректной работе распараллеливание не всегда даст желаемый результат. Команда на распараллеливание является очень ресурсоемкой и ей необходимо пользоваться осмотрительно. При малых размерах матриц оптимизация не даст желаемого ускорения, а в некоторых случаях возможно даже замедление, по сравнению с непараллельной версией программы. Исходя из этого, следует отметить, что распараллеливать необходимо только самые вычислительно емкие отрезки программы. Например, для метода вращений Якоби это перемножение матриц.

В [3] отмечается, что практические задачи характеризуются параметром масштаба, связанным с объемом входных данных (для задач МКЭ таким параметром масштаба служит количество КЭ). Показанный рост ускорения означает, что на реально используемом значении параметра масштаба – для реально встречающихся задач это десятки и сотни тысяч КЭ – ускорение будет еще больше.

ВЫВОДЫ

В работе рассмотрен простейший вариант оптимизации решения задач МКЭ путем распараллеливания пользовательской программы на языке C++ при помощи технологии OpenMP. Произведена оценка получаемого ускорения работы программы. Показано, что программное распараллеливание дает положительный эффект независимо от используемых аппаратных средств.

Литература

1. *Воеводин В.В., Воеводин В.В.* Параллельные вычисления. – СПб.: БХВ – Петербург, 2002.
2. *Зенкевич О.* Метод конечных элементов в технике. – М.: «Мир», 1975.
3. *Карпов В.Е.* Введение в распараллеливание алгоритмов и программ // Компьютерные исследования и моделирование. – 2010. – Т. 2. – № 3 – С. 231–272.
4. *Уилкинсон Д.Х.* Алгебраическая проблема собственных значений. – М.: «Наука», 1970.