УДК 519.853.4

Горчаков А.Ю.

Федеральный исследовательский центр «Информатика и управление» РАН, г. Москва, Россия

ИСПОЛЬЗОВАНИЕ ВОЗМОЖНОСТЕЙ OPENMP ДЛЯ УСКОРЕНИЯ РАСЧЕТОВ В ПАКЕТЕ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ADEPT*

Аннотация

В данной работе приводится пример использования возможностей OpenMP Application Programming Interface для ускорения расчетов матрицы первых производных вектор-функции (матрицы Якоби) при решении задач о наименьших квадратах методом Левенберга-Марквардта. Вычисление матрицы Якоби производится с помощью методологии быстрого автоматического дифференцирования, при этом применяется пакет прикладного программного обеспечения Adept версии 1.0. Для ускорения расчетов использовано директивное многопоточное программирование с динамическим распределением вычислений между потоками и технология SIMD (Single Instruction Multiple Data) – принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных. Вышеупомянутые технологии позволили в полной мере задействовать особенности архитектуры современных процессоров Intel, такие как многоядерность/многопоточность, расширение системы команд микропроцессоров Intel/AMD – Advanced Vector Extensions (AVX/AVX2), обрабатывающее данные в формате с плавающей запятой в группах длиной 256 бит, и FMA (Fused Multiply-Add) – технологию, предназначенную для выполнения совмещенной операции умножения-сложения. В результате, при минимальных трудозатратах, время вычисления матрицы первых производных вектор-функции на четырех ядерном процессоре удалось ускорить в 12 раз, по сравнению с вычислениями в однопоточном варианте.

Ключевые слова

OpenMP; SIMD; матрица Якоби; задача о наименьших квадратах; метод Левенберга-Марквардта.

Gorchakov A.Y.

Federal Research Center Computer Science and Control of the Russian Academy of Sciences, Moscow, Russia

USING THE OPENMP CAPABILITIES TO SPEED UP THE CALCULATIONS IN THE ADEPT SOFTWARE PACKAGE

Abstract

In this paper, an example is given of using the OpenMP Application Programming Interface to accelerate the calculation of the matrix of the first derivatives of a vector function (the Jacobi matrix) in the solution of least-squares problems by the Levenberg-Marquardt method. The calculation of the Jacobi matrix is produced using the methodology of fast automatic differentiation, the use of application software package Adept 1.0. To speed up the calculations, we used directive multithreaded programming with the dynamic distribution of computations between threads and SIMD (Single Instruction Multiple Data) technology, the principle of computer computation, which allows to provide parallelism at the data level. The aforementioned technologies allowed to fully exploit the features of the architecture of modern Intel processors, such as multi-core / multithreading, the expansion of the command system of microprocessors Intel / AMD – Advanced Vector Extensions (AVX / AVX2), processing data in floating point format in groups of 256 bits, and FMA (Fused Multiply-Add) – a technology designed to perform a combined multiply-add operation. As a result, with a minimum

* **Труды II Международной научной конференции «Конвергентные когнитивно-информационные технологии» (Convergent'2017), Москва, 24-26 ноября, 2017**

Proceedings of the II International scientific conference "Convergent cognitive information technologies" (Convergent'2017), Moscow, Russia, November 24-26, 2017

of labor, the time for calculating the matrix of the first derivatives of the vector function on four core processors was 12 times faster, compared to calculations in a single-threaded version.

Keywords

OpenMP; SIMD; the Jacobi matrix; the least-squares problem; the Levenberg-Marquardt method.

Введение

Среди задач безусловной оптимизации часто выделяют задачи минимизации функций F(x) вида:

$$F(x) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(x) = \frac{1}{2}\|f(x)\|_2{}^2 \rightarrow \min, \quad (1)$$

где *f(x)* – нелинейная вектор функция с *m* компонентами $f_i(x)$, а $x \in R^n$, $\|f(x)\|_2$- евклидова норма.

Задачи подобного типа возникают при математическом моделировании физических процессов, например, задача об оптимальном нагреве стержня [1], обратная коэффициентная задача для нестационарного уравнения теплопроводности [2], многие задачи машинного обучения.

Хотя для решения задач (1) можно использовать универсальные методы, более эффективным будет применение специальных алгоритмов, разработанных именно для задач о наименьших квадратах. Примерами таких алгоритмов являются метод Гаусса-Ньютона и Левенберга-Марквардта [3]. Оба метода используют особую структуру градиента функции и её матрицы Гессе.

Пусть *J(x)* матрица Якоби для *f(x)* и $H_i(x)$ матрица Гессе для $f_i(x)$. Тогда градиент *g(x)* и матрица Гессе *H(x)* для *F(x)* можно записать как:

$$g(x) = J(x)^T f(x), \quad (2)$$

$$H(x) = J(x)^T J(x) + Q(x), \quad (3)$$

где $Q(x) = \sum_{i=1}^{m} f_i(x)H_i(x)$. Сделав предположения о том, что слагаемое $J(x)^T J(x)$ доминирует над *Q(x)*, получаем следующие итерационные схемы методов.

Для метода Гаусса-Ньютона:

$$x_{k+1} = x_k + \alpha_k p_k, \quad (4)$$

где $0 < \alpha_k \leq 1$ – шаг метода, p_k- направление поиска, определяемое из решения системы уравнений

$$J_k^T J_k p_k = -J_k^T f_k\,. \quad (5)$$

Для метода Левенберга-Марквардта:

$$x_{k+1} = x_k + p_k, \quad (6)$$

где I – единичная матрица, λ_k – некоторая неотрицательная константа, своя для каждого шага, p_k - направление поиска, определяемое из решения системы уравнений

$$(J_k^T J_k + \lambda_k I)p_k = -J_k^T f_k. \quad (7)$$

Оба метода, при определенных условиях, могут достигать квадратичной скорости сходимости, хотя при расчете используются только первые производные (матрица Якоби). Следует отметить, что с развитием методологии быстрого автоматического дифференцирования (БАД) [4] скорость расчета матрицы Гессе скалярной функции *F(x)*, при $n \leq m$ приблизилась к скорости расчета матрицы Якоби вектор функции *f(x)*, и преимущества по сравнению с ньютоновскими методами существенно сократились.

Тем не менее, учитывая особенности вычисления матрицы Якоби и некоторые особенности задач, можно ускорить расчетные схемы методов Гаусса-Ньютона и Левенберга-Марквардта.

В данной статье рассматриваются способы ускорения расчетов матрицы Якоби, с помощью открытого стандарта для распараллеливания программ OpenMP. Рассмотрение производится на примере модельной задачи восстановления начальных данных для уравнения переноса. Для вычисления матрицы Якоби применен пакет прикладного программного обеспечения Adept [5], для решения систем линейных уравнений (7) использовался cLapack [6].

Методология быстрого автоматического дифференцирования

Пусть $z \in R^n$ и $u \in R^r$ векторы. Дифференцируемые функции *W(z, u)* и *Φ(z, u)* определяют отображения $W: R^n \times R^r \rightarrow R^1$, $\Phi: R^n \times R^r \rightarrow R^r$. Вектора z и u удовлетворяют следующей системе из n нелинейных скалярных уравнений *Φ(z, u)* = 0. Если матрица $\Phi_z^T(z, u)$ невырожденна, то сложная функция *Ω(u) = W(z(u), u)* дифференцируема и ее градиент относительно переменных u вычисляется по формуле:

$$\frac{d\Omega}{\mathrm{du}} = W_u(\mathrm{z(u)},\mathrm{u}) + \Phi_u^T(z(u),u)\cdot p, \quad (8)$$

где вектор $p \in R^n$ находится из решения линейной алгебраической системы:

$$W_z(\mathrm{z},\mathrm{u}) + \Phi_z^T(z,u)\cdot p = 0, \quad (9)$$

Здесь и далее индекс T обозначает транспонирование, нижние индексы z, u обозначают частные производные функций по векторам z и u: $W_u = \frac{\partial W}{\partial u}, W_z = \frac{\partial W}{\partial z}, \Phi_u^T = \frac{\partial \Phi^T}{\partial u}, \Phi_z^T = \frac{\partial \Phi^T}{\partial z}$, так же будем обозначать i-е, j-е компоненты векторов z и u как z_i, z_j, u_i, u_j.

Предположим, что каждый компонент вектора z_i последовательно выражается только через компоненты вектора u и предыдущие z_j, т.е. $1 \le j < i$., тогда формулы (8)-(9) можно записать в следующем виде:

$$p_i = \sum_{j=i+1}^{n} \Phi_{z_j}^T(z,u)p_j + W_{z_i}(\mathrm{z},\mathrm{u}), \quad (10)$$

$$\frac{d\Omega}{\mathrm{du}_i} = W_{u_i}(\mathrm{z},\mathrm{u}) + \sum_{j=1}^{n} \Phi_{u_j}^T(z,u)\cdot p_j \ . \quad (11)$$

В работе [4] дана оценка временной сложности вычисления градиента $T_g/T_0 \le 3$, где T_0 полное время, требуемое для вычисления значения функции, T_g дополнительное время, требуемое для вычисления всех частных производных функции; оценка дана без учета времени, необходимого для работы с памятью компьютера. Напомним, что в случае численного дифференцирования для расчета градиента (и матрицы Якоби) потребуется времени не менее $(1+n)T_0$, где n число необходимых нам частных производных. При расчете матрицы Якоби с помощью программного пакета Adept, для каждой строки запускается процедура расчета градиента $f_i(x)$, т.е. теоретическая оценка временной сложности вычисления $T_J/T_0 \le$ 3m, где T_J время вычисления матрицы Якоби.

Постановка модельной задачи

Рассмотрим задачу восстановления начальных данных для одномерного уравнения переноса пассивной примеси в жидкости движущейся с постоянным значением скорости:

$$\frac{\partial \varphi}{\partial t} + c\frac{\partial \varphi}{\partial x} = 0, c = 1, \quad (x,t) \in Q, \quad (12)$$

$$\varphi(x,0) = w_1(x), \qquad x \in [0,1], \quad (13)$$

$$\varphi(0,t) = w_2(t), \qquad t \in (0,1]. \quad (14)$$

Здесь x – пространственная координата; t – время; c – скорость движения жидкости; $\varphi(x,t)$ – концентрация примеси в точке x в момент времени t; прямоугольник $Q = (0,1)\times(0,1)$; $w_2(x)$ – заданная функция, определяющая концентрацию примеси в точке $x=0$; $w_1(x)$ – искомая функция, определяющая концентрацию примеси в начальный момент времени. Пусть на прямой $x+t=1$ задана функция $\bar{\varphi}(x,t)$, где $(x,t \in Q)$. Ее можно рассматривать как концентрацию примеси, полученную в результате некоторых экспериментальных исследований. Поставим следующую задачу: требуется определить оптимальное управление $u_*(x) = w_{1_*}(x)$ и соответствующее решение $\varphi_*(x,t)$ системы (12)-(14), при котором интегральный функционал

$$W(u) = \frac{1}{2}\int_0^1 [\varphi(x,1-x) - \bar{\varphi}(x,1-x)]^2\,dx \quad (15)$$

достигал бы минимально возможного значения.

Задачи, подобные этой, обычно решаются численно с помощью некоторого метода спуска, который требует знания градиента функционала (15).

Перейдем сразу же к дискретному варианту системы (12)-(14). Для этого покроем прямоугольник Q регулярной расчетной сеткой с шагами $h=1/J$ и $\tau=1/N$, где N – количество интервалов по времени, а J – количество интервалов по оси x (для удобства расчетов возьмем N кратным J, т.е. N=k*J). Используем хорошо известную схему «уголок», аппроксимирующую уравнение (12) на трехточечном шаблоне с первым порядком точности:

$$\frac{\varphi_j^{n+1} - \varphi_j^n}{\tau} + c\frac{\varphi_j^n - \varphi_{j-1}^n}{h} = 0\,, \quad (16)$$

$$\varphi_j^0 = w_{1_j} = u_j, \quad (17)$$

$$\varphi_0^n = {w_2}^n, \quad (18)$$

$$W(u) = \frac{1}{2}\sum_{j=1}^{J}(\varphi_j^{N-k*j} - \bar{\varphi}_j^{N-k*j})^2, \quad (19)$$

где

$$\varphi_j^n = \varphi(x_j, t_n), \bar{\varphi}_j^n = \bar{\varphi}(x_j, t_n), x_j = jh, t_n = n\tau,$$
$$\varphi_j^0 = w_{1_j} = u_j = u(x_j), \varphi_0^n = {w_2}^n = w_2(t_n),$$
$$0 \le j \le J, 0 \le n \le N.$$

Требуется найти вектор u, при котором сложная дискретизированная функция W(u) принимает минимальное значение.

Векторизация и многопоточность

Архитектура универсальных микропроцессоров постоянно развивается. Например, в процессоры Intel, вслед за многопоточностью и многоядерностью последовательно были добавлены технологии SIMD (англ. single instruction, multiply data – одиночный поток команд, множественный поток данных), такие как MMX, SSE, SSE2, SSE3, SSSE3, SSE4.x, AVX, AVX2, AVX-512.

Не вдаваясь в подробности скажем, что технология AVX2 (поддерживается с 2013 года в четвертом поколении процессоров Intel Core – Haswell) работает с 256 битными регистрами и позволяет одновременно обрабатывать 4 числа с двойной точностью (или 8 чисел с одинарной точностью). Так же поддерживается FMA3 (англ. Fused Multiply-Add, умножение-сложение с однократным округлением).

Вслед за изменением архитектуры микропроцессоров возникает необходимость модифицировать прикладное программное обеспечение, с целью адаптации его для более эффективного использования имеющихся вычислительных мощностей.

Причем, хотелось бы, на данную модификацию тратить минимум усилий. Вполне естественным вариантом ускорить выполнение прикладной программы, является использование возможностей компилятора. Например, используя известный компилятор gcc, можно векторизовать циклы простым указанием опций компиляции *–O3, -Ofast*, Данный способ достаточно эффективен, но распознавание циклов, которые можно векторизовать, происходит не во всех случаях. Помочь компилятору можно с помощью директив OpenMP [7], с их же помощью можно сделать программу многопоточной.

Приведем, очень кратко без полного описания синтаксиса, полезные директивы и опции компилятора gcc (более полное описание можно найти в [7], [8]):

#pragma omp parallel – с этой фундаментальной конструкции начинается параллельное исполнение программы.

#pragma omp for – данная конструкция указывает компилятору на необходимость многопоточного исполнения цикла. По умолчанию компилятор заранее разбивает цикл на *n* статических групп и формирует *n* потоков исполнения, где *n* – количество логических процессоров в системе (или число, указанное пользователем).

#pragma omp for schedule(dynamic) – разновидность предыдущей конструкции, указывающая компилятору, что разбиение цикла будет происходить динамически, в процессе выполнения. Если итерации цикла имеют различное время исполнения, например, 1-я итерация – 1 секунда, 2-я итерация 2 секунды, 3-я – 3 секунды и т.д., то использование данной конструкции предпочтительнее.

#pragma omp simd – конструкция позволяет векторизовать цикл.

#pragma omp simd reduction(reduction-identifier:list) – аналогично предыдущей конструкции, дополнительно производится операция редукции *(+, -, *, &, |, ^, &&, ||, max, min*) над указанным списком переменных.

-ftree-vectorizer-verbose=3 -fopt-info-vec-all=rezv.txt – ключи компилятора gcc выводящие в файл «rezv.txt» информацию о векторизации циклов.

Ускорение расчетов для метода Левенберга-Марквардта

Приведем описание использованного в численных экспериментах метода:

1. Положить k=0, $\lambda_0 = 0.01, \varepsilon = 10^{-20}$
2. Вычислить $F_k; f_k$
3. Вычислить J_k
4. Вычислить $J_k^T J_k$ и $-J_k^T f_k$
5. Сложить $\lambda_k I$ с $J_k^T J_k$
6. Найти p_k, решив систему уравнений (7)
7. Положить $u_{k+1} = u_k + p_k$
8. Вычислить $F_{k+1}; f_{k+1}$
9. Если $F_{k+1} < \epsilon$, то завершить работу.
10. Если $F_{k+1} < F_k$, то положить
$\lambda_{k+1} = 0.1\lambda_k, k = k + 1$, и перейти к шагу 3.
11. Если $\lambda_k > 10^{20}$, то завершить работу.
12. Если $F_{k+1} \ge F_k$, то положить $\lambda_{k+1} = 10\lambda_k$, и перейти к шагу 5.

Проведем расчеты в однопоточном и многопоточном режиме с различными флагами оптимизации.

Таблица 1. Сходимость метода Левенберга-Марквардта

Шаг	Значение найденного минимума
Начальная точка	1.430922e-01
1	5.963379e-05
2	2.654437e-10
3	1.241797e-17
4	3.839771e-25

Из таблицы 1 видно, что скорость сходимости метода Левенберга-Марквардта достаточна для того, чтобы решить задачу за 4 итерации. При этом функция вычисляется 5 раз, матрица Якоби 4 раза и 4 раза решается система уравнений (7).

Таблица 2. Время вычислений в зависимости от флагов оптимизации в однопоточном режиме

Шаг/Время сек	-O2	-O3	-Ofast
Вычисление функции	0.77	0.75	0.46
Вычисление матрицы Якоби	199.96	148.62	76.89
Вычисление коэффициентов системы уравнений (7)	18.15	18.17	8.49
Решение системы уравнений (7)	11.50	11.57	7.30
Всего	230.38	179.11	93.14

Таблица 3. Время вычислений в зависимости от флагов оптимизации в многопоточном режиме

Шаг/Время сек	-O2	-O3	-Ofast
Вычисление функции	0.75	0.74	0.46
Вычисление матрицы Якоби	50.47	44.38	43.64
Вычисление коэффициентов системы уравнений (7)	6.69	6.82	4.86
Решение системы уравнений (7)	11.56	11.56	7.21
Всего	69.48	63.50	56.18

Из таблиц 2 и 3 видно, что основная часть времени (70%-80%) уходит на расчет матрицы Якоби. При этом только за счет флагов оптимизации можно ускорить расчеты примерно в 2.5 раза. Если добавить к этому распараллеливание на 8 потоков (по количеству логических ядер процессора), то расчеты будут ускорены в 4 раза. Причем расчет матрицы Якоби ускоряется в 4.5 раз. Подобрав параметр (*ADEPT_MULTIPASS_SIZE*), отвечающий за количество столбцов матрицы Якоби, рассчитываемых одновременно в одном потоке можно снизить время расчета до 28 секунд.

Перейдем к анализу кода пакета Adept. Матрица Якоби рассчитывается с помощью функции:

Stack::jacobian_reverse_openmp(Real jacobian_out)*

функция устроена следующим образом – матрица Якоби разбивается по столбцам на блоки размера *ADEPT_MULTIPASS_SIZE*, при этом размер последнего блока может быть меньше этого параметра. Директивой *#pragma omp parallel* объявляется параллельный регион программы. Цикл по блокам разбивается на потоки с помощью директивы OpenMP – *#pragma omp for*. Как было сказано ранее статическое разбиение цикла не всегда является эффективным, и замена директивы на *#pragma omp for schedule(dynamic* позволяет получить ускорение расчетов порядка 30%.

Таблица 4. Время вычислений при использовании #pragma omp for schedule(dynamic)

Шаг/Время сек	-O2	-O3	-Ofast
Вычисление функции	0.76	0.75	0.46
Вычисление матрицы Якоби	38.38	32.86	29.17
Вычисление коэффициентов системы уравнений (7)	3.38	3.39	1.95
Решение системы уравнений (7)	11.68	11.57	7.20
Всего	54.21	48.57	38.77

Дальнейшее рассмотрение функции *Stack::jacobian_reverse_openmp(Real* jacobian_out)* показывает, что внутри основного цикла широко используется конструкции:

а) *for(int i = 0; i < block_size; i++)*, где *block_size* – размер блока

б) *int n_non_zero = 0;*

for (int i = 0; i < block_size; i++) {

```
    ...
    if (a[i] != 0.0) n_non_zero = 1;
  }
```

Используя ключи компилятора *-ftree-vectorizer-verbose=3 -fopt-info-vec-all=rezv.txt,* можно увидеть, что векторизация данных циклов не производится. То есть преимущества современной архитектуры процессора не используются.

Заменим данные конструкции на

а)
```
if(flag_extra)
  for (int i = 0; i < n_extra; i++)
    ...
 else
#pragma omp simd
  for (int i = 0; i < ADEPT_MULTIPASS_SIZE; i++)
    ...
```

б)
```
int n_non_zero = 0;
    if(flag_extra)
      for (int i = 0; i < n_extra; i++) {
      ...
      if(a[i] != 0) n_non_zero = 1;
      }
    else
#pragma omp simd reduction(|:n_non_zero)
      for (int i = 0; i < ADEPT_MULTIPASS_SIZE; i++) {
      ...
      n_non_zero |= (a[i] != 0);
      }
```

Здесь переменная цикла для большинства блоков изменяется от 0 до фиксированного значения *ADEPT_MULTIPASS_SIZE,* и включена принудительная векторизация директивой *#pragma omp simd.* В случае б) необходимо, кроме этого, выполнять редукцию переменной *n_non_zero.*

В результате, после подбора параметра *ADEPT_MULTIPASS_SIZE (рис.1),* получаем ускорение расчетов почти в 2 раза.

Таблица 5. Время вычислений при использовании SIMD

Шаг/Время сек	-O2	-O3	-Ofast
Вычисление функции	0.75	0.75	0.46
Вычисление матрицы Якоби	24.73	16.35	15.62
Вычисление коэффициентов системы уравнений (7)	3.44	3.46	1.92
Решение системы уравнений (7)	11.76	11.70	7.25
Всего	40.68	32.26	25.26

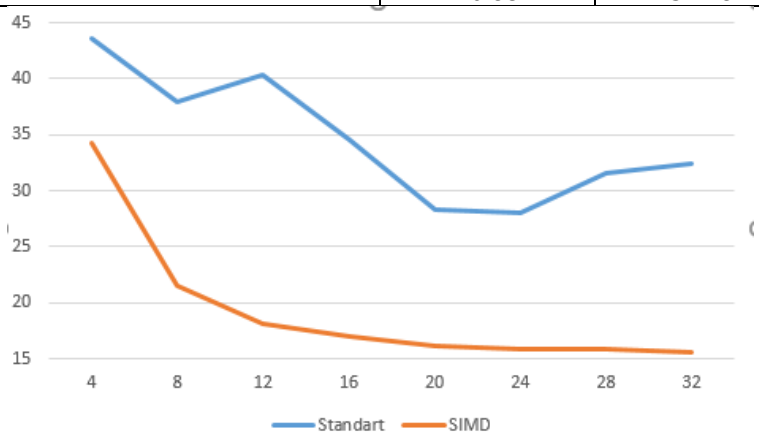

Рис.1 Время расчетов матрицы Якоби в зависимости от ADEPT_MULTIPASS_SIZE

Заключение

Исследование показало, что, используя технологии директивного многопоточного программирования, и возможности компилятора, можно ускорить расчет матрицы Якоби более чем в 12 раз. Подчеркнем на

процессоре, имеющем 8 логических ядер. При этом трудозатраты по модификации пакета прикладного программного обеспечения минимальны и ограничиваются использованием ключей компилятора и добавлением/изменением нескольких директив OpenMP.

В работе не был затронут вопрос распараллеливания методов решения систем линейных уравнений. Пути ускорения могут быть следующие – использование параллельных версий пакета LaPack, применение параллельных версий метода сопряженных градиентов. В ряде случаев будет эффективным использование двойственных постановок задач квадратичного программирования.

Благодарности

Работа выполнена при поддержке РФФИ, проект 16-07-00458.

Литература

1. Евтушенко Ю. Г., Зубов В. И. Об обобщенной методологии быстрого автоматического дифференцирования. Журнал вычислительной математики и математической физики, 2016, vol. 56 (11), pp. 1847-1862.
2. Зубов В. И. Применение методологии быстрого автоматического дифференцирования к решению обратной коэффициентной задачи для уравнения теплопроводности. Журнал вычислительной математики и математической физики, 2016, vol. 56 (10), pp 1760-1774.
3. Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. – 1985.
4. Евтушенко Ю. Г. Оптимизация и быстрое автоматическое дифференцирование //М.: Научное издание ВЦ РАН. – 2013.
5. Hogan R. J. Fast reverse-mode automatic differentiation using expression templates in C++ //ACM Transactions on Mathematical Software (TOMS). – 2014. – Т. 40. – №. 4. – С. 26.
6. Anderson E. et al. LAPACK Users' guide. – Society for Industrial and Applied Mathematics, 1999
7. OpenMP 4.5 Complete Specifications (Nov 2015)
8. Лупин С. А., Посыпкин М. А. Технологии параллельного программирования: Учеб. Пос. Сер. Высш. образ-ние. М.: Форум Инфра-М. – 2008, vol. 208.

References

1. Evtushenko Ju. G., Zubov V. I. Ob obobshhennoj metodologii bystrogo avtomaticheskogo differencirovanija. Zhurnal vychislitel'noj matematiki i matematicheskoj fiziki, 2016, vol. 56 (11), pp. 1847-1862.
2. Zubov V. I. Primenenie metodologii bystrogo avtomaticheskogo differencirovanija k resheniju obratnoj kojefficientnoj zadachi dlja uravnenija teploprovodnosti. Zhurnal vychislitel'noj matematiki i matematicheskoj fiziki, 2016, vol. 56 (10), pp 1760-1774.
3. Gill F., Mjurrej U., Rajt M. Prakticheskaja optimizacija. – 1985.
4. Evtushenko Ju. G. Optimizacija i bystroe avtomaticheskoe differencirovanie //M.: Nauchnoe izdanie VC RAN. – 2013.
5. Hogan R. J. Fast reverse-mode automatic differentiation using expression templates in C++ //ACM Transactions on Mathematical Software (TOMS). – 2014. – V. 40. – N. 4. – p. 26.
6. Anderson E. et al. LAPACK Users' guide. – Society for Industrial and Applied Mathematics, 1999
7. OpenMP 4.5 Complete Specifications (Nov 2015)
8. Lupin S. A., Posypkin M. A. Tehnologii parallel'nogo programmirovanija: Ucheb. Pos. Ser. Vyssh. obraz-nie. M.: Forum Infra-M. – 2008, vol. 208.

Сведения об авторе:

Горчаков Андрей Юрьевич, кандидат физико-математичесикх наук, ведущий математик отдела прикладных проблем оптимизации Вычислительного центра им. А.А. Дородницына, Федеральный исследовательский центр «Информатика и управление» Российской академии наук, andrgor12@gmail.com

Note on the author:

Gorchakov Andrei Yu., Candidate of Physical and Mathematical Sciences, Leading mathematician, Dorodnicyn Computing Centre, Federal Research Center Computer Science and Control of the Russian Academy of Sciences, andrgor12@gmail.com