УДК 004.42

Р. К. Кудерметов, канд. техн. наук

МНОГОПОТОЧНАЯ РЕАЛИЗАЦИЯ ЧЕТЫРЕХТОЧЕЧНОГО БЛОЧНОГО ОДНОШАГОВОГО МЕТОДА РЕШЕНИЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Аннотация. *Рассмотрена многопоточная реализация на языке Java четырехточечного блочного одношагового метода интегрирования обыкновенных дифференциальных уравнений. Предложен алгоритм реализации метода с помощью четырех потоков. В качестве механизма обмена данными между потоками использованы блокирующие очереди класса LinkedBlockingQueue. Приведены примеры задач для оценки правильности и точности метода. Обсуждаются условия, при которых многопоточная реализация данного метода становится эффективной.*

Ключевые слова: *численное интегрирование, многопоточность, параллельные вычисления, блокирующая очередь, многоядерные процессоры, гиперпоточность, ускорение, закон Амдала*

R. K. Kudermetov, PhD.

MULTI-THREADED REALIZATION OF THE FOUR-POINT BLOCK ONE-STEP METHOD FOR SOLVING DIFFERENTIAL EQUATIONS

Abstract. *The multi-threaded realization of the four-point block one-step method for solving ordinary differential equations is considered. The algorithm of the method implementation by four threads is proposed. For data exchanging between threads the blocking queues of Java's LinkedBlockingQueue library class are used. The conditions under which the multi-threaded realization of this method becomes effective are discussed.*

Keywords: *numerical integration, multi-threading, parallel computing, blocking queue, multi-core processors, hyper-threading, speed-up, Amdahl's Low*

Р. К. Кудерметов, канд. техн. наук

БАГАТОПОТОКОВА РЕАЛІЗАЦІЯ ЧОТИРЬОХТОЧКОВОГО БЛОЧНОГО ОДНОКРОКОВОГО МЕТОДУ РОЗВ'ЯЗАННЯ ДИФЕРЕНЦІЙНИХ РІВНЯНЬ

Анотація. *Розглянуто багатопотокову реалізацію мовою Java чотирьохточкового блочного однокрокового методу інтегрування звичайних диференційних рівнянь. Запропоновано алгоритм реалізації методу за допомогою чотирьох потоків. Як механізм обміну даними між потоками використано блокуючі черги класа LinkedBlockingQueue. Наведено приклади задач для оцінки правильності і точності методу. Обговорюються умови, за яких багатопотокова реалізація даного методу стає ефективною.*

Ключові слова: *чисельне інтегрування, багатопоточність, паралельні обчислення, блокуюча черга, багатоядерні процесори, гіперпотоковість, прискорення, закон Амдала*

Введение

В настоящее время большинство компьютеров общего назначения содержит двух- и четырехъядерные процессоры и наблюдается тенденция дальнейшего роста числа ядер. Эффективное их использование достигается методами параллельного и многопоточного программирования [1 – 2].

Существует множество языков, технологий, моделей параллелизма, таких как MPI, Erlang, Kilim (Java), GPars (Groovy), в которых обмен данными между потоками и процессами осуществляется с помощью сообщений, API Win32, .NET, POSIX Pthreads, OpenMP, Java, в которых обмен данными между потоками выполняется через разделяемую память. Здесь, говоря о многопоточности, мы имеем ввиду пользовательский уровень многопоточности, в отличие от потоков уровня ядра.

При компьютерном моделировании систем автоматического управления динамику объектов управления обычно моделируют, решая обыкновенные дифференциальные уравнения с помощью одношаговых методов интегрирования, таких как методы Рунге-Кутты, блочные одношаговые методы и др. Для этих методов существуют и продолжают разрабатываться последовательные и параллельные алгоритмы реализации. Разработка реализаций этих алгоритмов с учетом современного инструментария многопоточного программирования и многоядерных и гиперпоточных архитектур процессоров может принести значительный выигрыш в производительности.

В данной работе рассматривается многопоточная реализация параллельного четы-

рехточечного блочного одношагового метода интегрирования основные характеристики (точность, устойчивость), которого наряду с блочными многошаговыми методами рассмотрены в [3].

Параллельные двух- и четырехточечные блочные методы интегрирования, благодаря возможности выполнять итерации приближения к решению интегрируемого уравнения в точках блока независимо, можно считать хорошо согласованными с архитектурами многоядерных процессоров и процессоров/ядер со встроенной гиперпоточностью. Используя модель «распараллеливания по заданиям» [2], вычисления в каждой точке блока метода интегрирования можно выполнять в отдельных потоках пользовательского уровня, которые планировщик Java-виртуальной машины (JVM) распределит на отдельные физические или логические ядра процессора.

Постановка задачи

Для численного решения обыкновенных дифференциальных уравнений в форме

$$\frac{dy}{dt} = f(t, y),\ y(t_0) = y_0 \qquad (1)$$

на равномерной сетке по аргументу t с шагом h блочными одношаговыми методами интегрирования разностные уравнения могут быть представлены как

$$\frac{y_{n,i} - y_{n,0}}{ih} = b_i F_{n,0} + \sum_{j=1}^{k} a_{i,j} F_{n,j}, \qquad (2)$$

где $i = \overline{1,k}$ - номер точки в блоке;

$n = 1, 2, \ldots$ - номер блока;

$F_{n,j} = f\left(t_n + jh, y(t_n + jh)\right)$.

Погрешность аппроксимации k-точечного блочного одношагового метода имеет порядок p, если выполняются условия [3]:

$$b_i + \sum_{j=1}^{k} a_{i,j} = 1,\ i = \overline{1,k}, \qquad (3)$$

$$\sum_{j=1}^{k} j^{m-1} a_{i,j} = \frac{i^{m-1}}{m},\ m = \overline{2,p}. \qquad (4)$$

Если принять $p = k + 1 = 5$, то, решая систему уравнений (3) и (4), найдем все коэффициенты b_i и $a_{i,j}$ для четырехточечного метода. Ниже вычисленные коэффициенты представлены в виде вектора и матрицы:

$$B = \left(\frac{251}{720} \quad \frac{29}{180} \quad \frac{9}{80} \quad \frac{7}{90}\right), \qquad (5)$$

$$A = \begin{pmatrix} \frac{323}{360} & -\frac{11}{30} & \frac{53}{360} & -\frac{19}{720} \\ \frac{31}{45} & \frac{2}{15} & \frac{1}{45} & -\frac{1}{180} \\ \frac{17}{40} & \frac{3}{10} & \frac{7}{40} & -\frac{1}{80} \\ \frac{16}{45} & \frac{2}{15} & \frac{16}{45} & \frac{7}{90} \end{pmatrix}. \qquad (6)$$

Наивысший порядок аппроксимации метода будет равен $k+1$, т. е. $O\left(h^5\right)$, а погрешность в точках блока будет определяться формулой [3]:

$$\varepsilon_{n,i} = \frac{h^{k+1}}{(k+1)!} y_{n,0}^{(k+2)} \left(\sum_{j=1}^{k} j^{k+1} a_{i,j} - \frac{i^{k+1}}{k+2} \right) + O\left(h^{k+2}\right),\ i = \overline{1,k}. \qquad (7)$$

Таким образом, погрешность четырехточечного блочного одношагового метода в точках блока будет иметь порядок $O\left(h^6\right)$. В [3] также показано, что шаг интегрирования необходимо выбирать согласно условию

$$h < \frac{1}{kL\|A\|}, \qquad (8)$$

где L - константа из условия Липшица $\left|f(t, y_1) - f(t, y_2)\right| \le L\left|y_1 - y_2\right|$, которому должна удовлетворять функция правой части уравнения (1) при любых t, y_1, y_2 из рассматриваемой области;

$\|A\| = 1{,}4375$ - норма матрицы коэффициентов A из (6).

Целью данной работы является разработка параллельной реализации четырехточечного блочного одношагового метода интегрирования для двух- и четырехъядерных процессоров с использованием потоков технологии Java.

Параллельный алгоритм метода

Для вычисления приближенных значений $y_{n,k}$ неизвестной y задачи Коши (1) в точках k каждого n-го блока по формуле (2) необходимо выполнить итерации с помощью следующих рекуррентных формул, полученных после подстановки в (2) коэффициентов (5) и (6) [3, 4]:

$$y_{n+k,0} = y_n + khF_n,\ k = \overline{1,4}\ , \qquad (9)$$

$$y_{n+1,s+1} = y_n + \frac{h}{720}\left(251F_n + 646F_{n+1,s} - 256F_{n+2,s} + 106F_{n+3,s} - 19F_{n+4,s}\right), \qquad (10)$$

$$y_{n+2,s+1} = y_n + \frac{h}{90}\left(29F_n + 124F_{n+1,s} + 2F_{n+2,s} + 4F_{n+3,s} - F_{n+4,s}\right), \qquad (11)$$

$$y_{n+3,s+1} = y_n + \frac{3h}{80}\left(9F_n + 34F_{n+1,s} + 24F_{n+2,s} + 14F_{n+3,s} - F_{n+4,s}\right), \qquad (12)$$

$$y_{n+4,s+1} = y_n + \frac{2h}{45}\left(7F_n + 32F_{n+1,s} + 12F_{n+2,s} + 32F_{n+3,s} + 7F_{n+4,s}\right), \qquad (13)$$

где $s = \overline{0,S}$ - номер итерации;

y_n - численная аппроксимация точного решения в точке n сетки по аргументу t;

$y_{n+k,s}$ - численная аппроксимация точного решения в k-й точке блока на итерации s;

$F_n = f(t_n, y_n)$ - значение функции $f(t, y)$ в начальной точке n-го блока;

$F_{n+k,s+1} = f\left(t_n + kh, y_{n+k,s}\right)$ - значение функции $f(t, y)$ в k-й точке n-го блока на итерации $s+1$.

Реализация метода организована следующим образом. Вычисления в каждой из четырех точек блока выполняют отдельные потоки, условно названные как Thread-1, Thread-2, Thread-3 и Thread-4. Алгоритмы вычислений для каждого из потоков приведены в алгоритмах 1, 2, 3.

Результатом последней итерации в потоке Thread-4 является приближенное решение (1) в последней, четвертой точке n-го блока на сетке с шагом h. Это же значение является начальным для вычисления в следующем, $n+1$-м блоке. Кроме того, после последней итерации в этом потоке выполняется увеличение текущего времени интегрирования на величину $4h$.

```
1   t1 ← t0
2   y1 ← y0
3   while t1 < T do //T – время решения
4       F0 ← f(t1, y1)
5       for i ← 2 to 4 do
6           F0,i ← F0
7           put(F0,i)
8       end for
9       u1 ← y1 + hF0
10      for j ← 0 to S do
11          F1 ← f(t1 + h, u1)
12          for i ← 2 to 4 do
13              F1,i ← F1
14              put(F1,i)
15              take(Fi,1)
16          end for
17          u1 ← y1 + h/720 (251F0 + 646F1 −
                256F2,1 + 106F3,1 − 19F4,1)
18      end for
19      take(t1)
20      take(y1)
21  end while
```

Алгоритм 1. Вычисления в потоке Thread-1

```
1   r ← 2
1´  r ← 3
2   tr ← t0
3   yr ← y0
4   while tr < T do
5       take(F0,r)
7       ur ← yr + rhF0,r
8       for j ← 0 to S do
9           Fr ← f(tr + rh, ur)
10          for i ← 1 to k do
11              if i ≠ r do
12                  Fr,i ← Fr
13                  put(Fr,i)
14                  take(Fi,r)
15              end if
16          end for
17          ur ← yr + h/90 (29F0,r + 124F1,r +
                2F2,r + 4F3,r − F4,r)
17´         ur ← yr + 3h/80 (9F0,r + 34F1,r +
```

$24F_{2,r} + 14F_{3,r} - F_{4,r}\big)$
18 **end for**
19 take(t_r)
20 take(y_r)
21 **end while**

Алгоритм 2. Вычисления в потоках Thread-2 и Thread-3. Шаги 1 и 17 выполняются потоком Thread-2, шаги 1´ и 17´ выполняются потоком Thread-3

1 $t_4 \leftarrow t_0$
2 $y_4 \leftarrow y_0$
3 **while** $t_4 < T$ **do**
4 take$(F_{0,4})$
5 $u_4 \leftarrow y_4 + 4hF_{0,4}$
6 **for** $j \leftarrow 0$ **to** S **do**
7 $F_4 \leftarrow f(t_4 + 4h, u_4)$
8 **for** $i \leftarrow 1$ **to** 3 **do**
9 $F_{4,i} \leftarrow F_4$
10 put$(F_{4,i})$
11 take$(F_{i,4})$
12 **end for**
13 $u_4 \leftarrow y_4 + \frac{2h}{45}\big(7F_{0,4} + 32F_{1,4} + 12F_{2,4} + 32F_{3,4} + 7F_4\big)$
14 **end for**
15 $t_4 \leftarrow t_4 + 4h$
16 $y_4 \leftarrow u_4$
17 **for** $i \leftarrow 1$ **to** 3 **do**
18 $t_i \leftarrow t_4$
19 $y_i \leftarrow y_4$
20 put(t_i)
21 put(y_i)
22 **end for**
23 **end while**

Алгоритм 3. Вычисления в потоке Thread-4

Для обмена результатами вычислений между потоками используются блокирующие очереди класса **LinkedBlockingQueue** пакета **java.util.concurrent** и его методы **put()**и **take()**. Метод **put()** записывает данные в блокирующую очередь, а метод **take()** извлекает данные из блокирующей очереди [5]. В программной реализации данного метода введены блокирующие очереди $F_{i,j}$, y_j, t_j, где $i = \overline{0,4}$ - индекс функции; $j = \overline{1,4}$ - номер потока.

Блокирующие очереди класса **LinkedBlockingQueue** организованы по принципу очереди FIFO. Блокирование работы потоков осуществляется при попытке записать данные в заполненную очередь или считать данные из пустой очереди.

Таким образом, данные, получаемые потоками из блокирующих очередей, всегда являются актуальными, но при этом неизбежны простои в работе потоков, связанные с ожиданиями освобождения и заполнения элементов очередей данными.

Для проверки точности метода и правильности его программной реализации выполним решения для следующих примеров.

1. Умеренно устойчивая задача [3]

$$\frac{dy}{dt} = -10(t-1)y,\ y(0) = 1,\ t = [0,2] \qquad (14)$$

имеет аналитическое решение $y(t) = e^{-5t(t-2)}$. Шаг интегрирования должен удовлетворять условию (8): $h < \frac{1}{4 \cdot 10 \cdot 1{,}4375} = 0{,}0174$.

2. Жесткая задача [6]

$$\frac{dy}{dt} = -9y,\ y(0) = e,\ t = [0,1] \qquad (15)$$

имеет аналитическое решение $y(t) = e^{1-9t}$.
Шаг интегрирования определим из условия

$$h < \frac{1}{4 \cdot 9 \cdot 1{,}4375} = 0{,}0193.$$

3. Задача [7]

$$\frac{dy}{dt} = \lambda(y - t) + 1,\ y(0) = 1,\ t = [0,1] \qquad (16)$$

с аналитическим решением $y(t) = e^{\lambda t} + t$ является жесткой для отрицательных λ. При $\lambda = -20$ шаг интегрирования определяется из условия $h < \frac{1}{4 \cdot 20 \cdot 1{,}4375} = 0{,}008695$.

Максимальные погрешности численного решения задач (14), (15) и (16) δ, полученные путем сравнения этих решений с аналитическими, при различных значениях шага интегрирования h и числа итераций S, приведены в таблицах 1; 2 и 3.

1. Погрешность решения задачи (14).

S	δ (h = 0,02)	δ (h = 1/64)	δ (h = 0,01)
5	$3{,}35\cdot10^{-4}$	$6{,}78\cdot10^{-5}$	$3{,}42\cdot10^{-6}$
6	$9{,}02\cdot10^{-6}$	$3{,}00\cdot10^{-6}$	$3{,}18\cdot10^{-7}$
7	$2{,}79\cdot10^{-5}$	$6{,}13\cdot10^{-6}$	$4{,}04\cdot10^{-7}$

2. Погрешность решения задачи (15)

S	δ (h = 0,02)	δ (h = 1/64)	δ (h = 0,01)
5	$3{,}66\cdot10^{-6}$	$5{,}50\cdot10^{-7}$	$1{,}70\cdot10^{-8}$
6	$6{,}00\cdot10^{-7}$	$1{,}20\cdot10^{-7}$	$7{,}24\cdot10^{-9}$
7	$4{,}50\cdot10^{-7}$	$1{,}00\cdot10^{-7}$	$6{,}82\cdot10^{-9}$

3. Погрешность решения задачи (16)

S	δ (h = 1/64)	δ (h = 1/128)	δ (h = 0,005)
5	$1{,}15\cdot10^{-4}$	$4{,}54\cdot10^{-7}$	$1{,}40\cdot10^{-8}$
6	$1{,}45\cdot10^{-5}$	$8{,}68\cdot10^{-8}$	$5{,}00\cdot10^{-9}$
7	$6{,}62\cdot10^{-6}$	$7{,}00\cdot10^{-8}$	$4{,}70\cdot10^{-9}$

Параллельные вычисления рассматриваемой реализации сопровождаются затратами времени на ожидания потоками высвобождения и заполнения блокирующих очередей. Предлагаемая реализация может быть эффективной в случае, если время, необходимое на вычисления правых частей уравнения (1), будет существенно превосходить время простоев потоков, а также затраты времени на выполнение операций метода интегрирования. Из формул метода (9) – (13) следует, что при однопоточной реализации метода для получения решения на одном блоке функцию правой части (1) необходимо вычислить $4S+1$ раз. Согласно алгоритмам 1; 2 и 3, в потоке Thread-1 вычисление функции правой части (1) выполняется $S+1$ раз, а в остальных потоках - это число равно S.

Следовательно, если суммарные затраты времени на вызовы и вычисления функции правой части много больше затрат времени на синхронизацию потоков и выполнение операций метода интегрирования, то ускорение параллельной реализации метода Sp можно оценить как отношение, числа вызовов функции правой части последовательной реализации метода, к числу таких вызовов при параллельной реализации:

$$Sp = (4S+1)/(S+1). \qquad (17)$$

Например, если $S = 6$, то на четырехъядерном процессоре максимальное ускорение при пренебрежении накладными расходами будет равно 3,7, на двухъядерном процессоре – 1,78. Такие же оценки можно получить, если воспользоваться известным законом Амдала [2; 8].

На рис. 1 приведены результаты экспериментального определения ускорения для четырехъядерного процессора AMD A6-3650 APU 2,60 GHz.

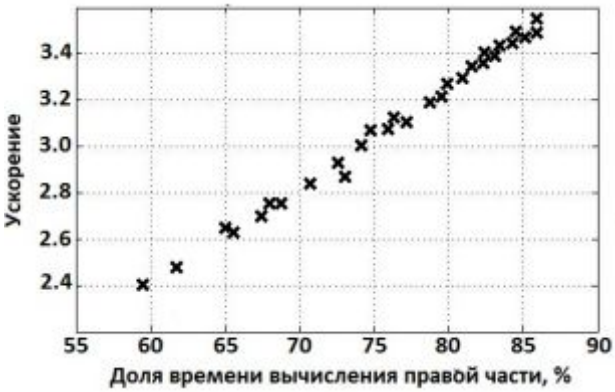

Рис. 1. Зависимость ускорения метода от доли времени вычисления функции правой части уравнения (1) при решении на четырехядерном процессоре и $S = 6$

Как видим из графика (рис. 1), при увеличении доли времени, затрачиваемой на вычисления правой части, от суммарного времени реализации метода, значение ускорения приближается к оценке (17).

На рис. 2 представлены результаты измерения ускорения реализации при использовании двухъядерного процессора со встроенной технологией гиперпоточности Intel® Core™ i3-2310M @ CPU 2.10 GHz. Теоретическое максимальное ускорение можно оценить с помощью формулы Амдала, описывающей типичное ускорение для приложений, выполняющихся на ядрах процессора, поддерживающих гиперпоточность [2]

$$S_{HT} = \frac{1}{\alpha + 0{,}67\dfrac{1-\alpha}{n} + H(n)}, \qquad (18)$$

где α - доля операций алгоритма, выполняющаяся на одном ядре;

n - количество логических ядер;

$H(n)$ - накладные расходы.

Приложение выполняется на двухъядерном процессоре с гиперпоточностью, то есть мы имеем четыре логических ядра ($n = 4$). Если пренебречь накладными расходами и за единицу принять число вызовов

функции правой части, которое при $S = 6$ равно 25 (алгоритмы 1, 2, 3), тогда $\alpha = 1/25$, так как в алгоритме 1, в отличие от остальных алгоритмов, на один вызов функции больше. Получим из (17):

$$S_{HT} = \frac{1}{0{,}04 + 0{,}67 \cdot 0{,}96/4} \approx 2{,}496 .$$

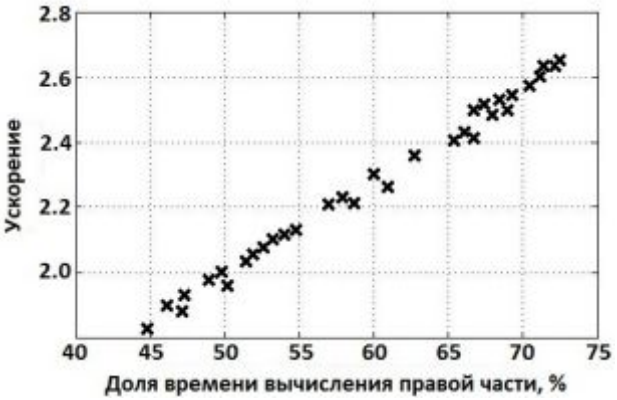

Рис. 2. Зависимость ускорения метода от доли времени вычисления функции правой части уравнения (1) при решении на двухядерном процессоре с гиперпоточностью

На рисунках 1 и 2 приведены только некоторые, статистически обработанные результаты оценки ускорения. Известно, что измерение производительности и интерпретация его результатов для приложений, написанных на динамически компилируемых языках, таких как Java - далеко не тривиальная задача [8; 9].

Существует множество причин, недетерминизма времени выполнения приложений на Java. Часто общераспространенные методики измерения производительности дают противоречивые результаты даже при повторных запусках одного и того же приложения. Наиболее распространенными источниками недетерминизма являются [10]: JIT-компилятор; невозможность по коду предсказать нужное число циклов для его тренировки («разогрев») на конкретном приложении; стратегии оптимизации и планирования работы потоков JVM; взаимное влияние процессов JIT-компилятора и Garber collection («сборки мусора»); другие внешние воздействия, например, прерывания операционной системы.

Выводы

Рассмотренная многопоточная параллельная Java-реализация четырехточечного блочного одношагового метода интегрирования обыкновенных дифференциальных уравнений может быть эффективной для решения практических задач на многоядерных процессорах и процессорах, использующих технологию гиперпоточности. Желаемое ускорение решения может быть достигнуто при условии, что затраты времени на синхронизацию потоков и выполнение операций метода интегрирования значительно меньше времени, необходимого для вычисления правых частей дифференциальных уравнений.

Список использованной литературы

1. Herlihy M., and Shavit N., (2008), The art of Multiprocessor Programming. *Burlington, MA, USA: Morgan Kaufmann Publishers Inc*, 508 p., ISBN 0123705916 9780123705914.

2. Эхтер Ш. Многоядерное программирование / Ш. Эхтер, Дж. Робертс. – СПб. : Питер, 2010. – 316 с.
- ISBN 978-5-388-00091-0.

3. Фельдман Л. П. Эффективные методы распараллеливания численного решения задачи Коши для обыкновенных дифференциальных уравнений / Л. П. Фельдман, О. А. Дмитриева. // Математическое моделирование. – М. : – 2001. – Вып. 13(7). – С. 66 – 72.

4. Системы параллельной обработки : под ред. Д. Ивенса. – М. : Мир, 1985. – 416 с.

5. Horstmann C. S., and Cornell G., (2004), Core Java, Volume II - Advanced Features, *7th ed. New Jersey: Prentice Hall*, 1024 p.
ISBN 0-13-111826-9.

6. Abubakar M. B., Ali M.B., and Muktar I.B., (2014), Formulation of 'Predictor-Corrector' Methods From 2-Step Hybrid Adams Methods for the Solution of Initial Value Problems of Ordinary Differential Equations, *International Journal of Engineering and Applied Sciences*, 5(3), pp. 9 – 13.

7. Muhammad R., and Yahaya Y.A., (2012), A Sixth Order Implicit Hybrid Backward Differentiation Formulae (HBDF) for Block Solution of Ordinary Differential Equations, *American Journal of Mathematics and Statistics*, 2(4), pp. 89 – 94, doi: 10.5923/j.ajms.20120204.04.

8. Popov G., Mastorakis N., and Mladenov V., (2010), Calculation of the Acceleration of Parallel Programs as a Function of the Number of Threads. *In: Proceeding ICCOMP'10 Proceedings of the 14th WSEAS International Conference on Computers*, Vol. II, pp. 411 – 414. ISBN: 978-960-474-213-4.

9. Goetz B, (2004), Java theory and Practice: Dynamic Compilation and Performance Measurement – The Perils of Benchmarking under Dynamic Compilation. IBM Developer Works. Available from: http://www.ibm.com/ developerworks/library/j-jtp12214/ (Accessed 16.11.2014).

10. Georges A., Buytaert D., Eeckhout L., (2007), Statistically Rigorous Java Performance Evaluation. *OOPSLA '07 Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications,* pp. 57 – 78.

Получено 20.11.2014

References

1. Herlihy M., and Shavit N., (2008), The art of Multiprocessor Programming. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 508 p. ISBN 0123705916 9780123705914. (In English).

2. Ekhter SH., and Roberts DZH., Mnogoyadernoe programmirovanie, [Multi-Core Programming: Increasing Performance through Software Multi-Threading], (2010), St. Peterburg, Russian Federation, *Piter Publ.,* 316 p. (In Russian).

3. Fel'dman L. P., and Dmitrieva O. A., Effektivnye metody rasparallelivaniya chislennogo resheniya zadachi Koshi dlya obyknovennykh differentsial'nykh uravnenij. [Effective Methods of Parallelization of the Numerical Solution of the Cauchy Problem for Ordinary Differential Equations], (2001), *Matematicheskoe Modelirovanie*, Moscow, Russian Federation, Vol. 13(7), pp. 66 – 72 (In Russian).

4. Ivens D. (ed.) Sistemy parallel'noi obrabotki [Parallel Processing Systems], (1985), Moscow, Russian Federation, *Mir Publ.,* 416 p. (In Russian).

5. Horstmann C.S., and Cornell G., (2004), Core Java, Volume II - Advanced Features. *7th ed. New Jersey: Prentice Hall,* 1024 p., ISBN 0-13-111826-9 (In English).

6. Abubakar M.B., Ali M.B., and Muktar I. B., (2014), Formulation of 'Predictor-Corrector' Methods From 2-Step Hybrid Adams Methods for the Solution of Initial Value Problems of Ordinary Differential Equations, *International Journal of Engineering and Applied Sciences*, 5(3), pp. 9 –13 (In English).

7. Muhammad R., and Yahaya Y.A., (2012), A Sixth Order Implicit Hybrid Backward Differentiation Formulae (HBDF) for Block Solution of Ordinary Differential Equations, *American Journal of Mathematics and Statistics*, 2(4), pp. 89 – 94, doi: 10.5923/j.ajms.20120204.04 (In English).

8. Popov G., Mastorakis N., and Mladenov V., (2010), Calculation of the Acceleration of Parallel Programs as a Function of the Number of Threads, In: *Proceeding ICCOMP'10 Proceedings of the 14th WSEAS international conference on Computers – Volume II, 2010,* pp. 411 – 414, ISBN: 978-960-474-213-4 (In English).

9. Goetz B, (2004), Java theory and Practice: Dynamic Compilation and Performance Measurement – The perils of Benchmarking under Dynamic Compilation. IBM Developer Works. Available from: http://www.ibm.com/ developerworks/ library/j-jtp12214/. (Accessed 16.11.14) (In English).

10. Georges A., Buytaert D., and Eeckhout L., (2007), Statistically Rigorous Java Performance Evaluation. In: *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications*, pp. 57 – 78 (In English).

Кудерметов
Равиль Камилович,
канд. техн. наук, зав. каф. компьютерных систем и сетей Запорожского нац. технического ун-та.
E-mail: krk@zntu.edu.ua